University of Alberta

A Performance Evaluation Framework of Multi-resolution Algorithms for Real-time Rendering

by

Ping Yuan ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Department of Computing Science

Edmonton, Alberta
Spring 2001

0-612-60363-6

[illegible]

Ping Yuan
Apt1703, 15 Brookbanks Dr.
North York, ON.
Canada, M3A 2T1

Date: Dec. 11th. 2000

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **A Performance Evaluation Framework of Multi-resolution Algorithms for Real-time Rendering** submitted by Ping Yuan in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

[illegible]
Dr. Mark W. Green

[illegible]
Dr. John Buchanan

[illegible]
Dr. Walter Bischof

Janelle Harms
Dr. Janelle Harms

[illegible]
Dr. Xiaoling Sun

[illegible]
per: Dr. Michael Zyda

Date: Dec. 11th. 2000

To my parents

Abstract

Many multi-resolution algorithms have been proposed to solve real-time rendering problems. However, the performance of these algorithms have not been evaluated, and there is no comparison between them. It is not clear how well these algorithms perform and under what conditions their performance is optimal. Thus the advances in multi-resolution algorithms can not be widely applied in real-time rendering applications.

This thesis presents a performance evaluation framework for multi-resolution algorithms for real-time rendering. It consists of a set of automatic performance measures and a standard real-time rendering testbed. This framework provides a unified system environment to automatically measure a range of multi-resolution algorithms while they perform various real-time rendering benchmarks. Experimental results show that the measured performance data are meaningful and consistent, and can form the basis for performance comparison of the measured algorithms. Several multi-resolution algorithms have also been evaluated using a prototype of the framework. The performance results help the user to compare their performance. These encouraging results demonstrate the potential of the performance evaluation framework, and indicate that it can ultimately promote the application of multi-resolution algorithms in real-time rendering systems.

Acknowledgements

I am deeply indebted to my parents, Rongxi Yuan and Peixia Zhao. Where I am today is in no small part due to their love and support.

I would like to express my sincere thanks to Dr. Mark Green, my supervisor for his encouragement and assistance throughout the course of this research. His many suggestions have been invaluable. I would also like to thank the rest of my thesis committee, Dr. John Buchanan, Dr. Walter Bischof, Dr. Janelle Harms, Dr. Xiaoling Sun, and Dr. Michael Zyda for their help.

Thanks go to the graphics research group at the University of Alberta for many interesting discussions. To University of Alberta, Faculty of Graduate Studies and Research, and Department of Computing Science, for technical and financial support.

To my friends Rona, Yuan, Yanping, Hongzhi, Nicolas, Paul, and Ruyam for their friendship and help.

To my sister, Fang, for her love and motivating me to graduate.

Finally, I would like to thank my husband, Dr. Qi Zhang, for his love, understanding and many insightful comments on this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Multi-resolution algorithms for real-time rendering

One of the challenges in the interactive 3D computer graphics field is real-time rendering. Given enough CPU-time, current computer graphics technologies can render very complex scenes and produce nearly photo-realistic pictures. However, for interactive 3D computer graphics, such as virtual reality applications and real-time CAD tasks, real-time response and natural motion is as important as, or even more important than, image realism to users [21]. For example, at 1 frame per second, an interactive 3D graphics application system is painful to use no matter how good the rendered image is [1]. In large-scale interactive 3D graphics applications, the required interactive frame rate and large, complex 3D models push the limits of graphics technology. Data management and programming decisions have to be made in trading off graphics rendering quality for interactive update rates. This has been called the real-time rendering problem. This problem exists in low end systems, where computer games and distributed virtual environments must often operate on systems where the available resources are highly constrained. It also arises at the high end as well, where realistic simulation and scientific visualization systems typically have object databases that far exceed the capacity of even the most powerful graphics workstations.

Many speed-up techniques for the graphics pipeline, such as Z-buffer, can help in certain situations. However most of the time, the real problem is that many complex details are not necessary for rendering a scene and also scene complexity varies from one frame to another, resulting in slow and fluctuating update rates. This problem can be solved if one is able to render the proper approximation of the original data set for each viewpoint and orientation in each frame. Multi-resolution algorithms are developed for this purpose. They are based on the observation that coarser models take less time to render and they appear similar to the original ones if used properly. They use several levels of detail of the original models to achieve maximum display efficiency during real-time rendering. A multi-resolution algorithm consists of two components. First, a multi-resolution modeling technique is needed to create, store and retrieve level of detail hierarchies to enable switching from one level of detail to another. Second, a run-time display mechanism for determining and rendering the best level of detail for models for each viewpoint and orientation. They work together to achieve the goal of maintaining a user required frame rate while preserving the image fidelity as much as possible.

1.2 Performance evaluation of multi-resolution algorithms

Dozens of multi-resolution algorithms have been proposed for real-time rendering. However, all the current algorithms produce image artifacts of some form, as the nature of the algorithms is to trade off rendering quality for frame rate. Some algorithms can not even guarantee an increased frame rate with these image artifacts, since their decision strategies are so expensive that they use up any gain in rendering time. Therefore, the performance of a multi-resolution algorithm is not obvious without thorough tests and careful measurements being done. Currently, the performance of these algorithms has not been evaluated thoroughly. Quite often the papers presenting the algorithms do not provide a thorough evaluation of their performance, leaving the reader with no idea of how they fit into the existing set of algorithms. In some cases the algorithms do not improve the rendering speed, and introduce artifacts that would not be present if they were not used in real-time rendering applications. Some algorithms were only tested under limited conditions, such as testing a algorithm while doing a fixed path navigation in a simple virtual environment. In addition, there is no comparison between the algorithms. This makes it difficult to determine how well these algorithms perform, which one is better than the others and under what conditions an algorithm's performance is optimal. Thus, the advances in real-time rendering algorithms can not be widely applied to interactive 3D computer graphics systems in the real world.

The reason for the limited performance evaluation of algorithms is that evaluating multi-resolution algorithms in real-time rendering systems is much more difficult than it looks. There is no previous experience on conducting broad tests of these algorithms. There are no performance metrics defined for multi-resolution algorithms and no clear definition of what tasks algorithms should carry out. There are many different hardware configurations and support software configurations involved for testing these algorithms and complex virtual environment models for testing are tedious to build, complicating the evaluation process. However, a fair and thorough performance evaluation is crucial to determine how well an algorithm works and how much it indeed helps in solving the real-time rendering problem.

In order to solve this problem, a standard performance evaluation system for multi-resolution algorithms is needed. Fair and thorough measurements of these algorithms are required. The major thrusts of this research are to investigate the performance evaluation techniques for multi-resolution algorithms in real-time rendering applications. A standard evaluation framework for evaluating algorithms is built upon them, so that a number of the existing real-time rendering algorithms can be evaluated efficiently. The measurements are consistent and meaningful, resulting in a relative ranking of their performance. The evaluations can be extended to new algorithms easily, so the new algorithms can be compared with existing ones.

1.3 Contributions

The primary contributions of this thesis are as follows:

- Key performance metrics are defined for multi-resolution algorithms in real-time rendering. The algorithms are expected to maintain constant frame rate that is above a certain threshold. They should preserve image fidelity in both the spatial and temporal domains. They should also keep preprocessing time and extra resource consumption to a minimum.

- Novel automatic performance measures of frame rate, spatial image fidelity, and temporal image fidelity are developed. These measures are able to compute how well the algorithms achieve the ideal situation. They are specific, realizable and easy to interpret.

- A standard real-time rendering testbed is developed. It is capable of loading and running various real-time rendering benchmarks based on virtual environments and navigation paths. It also provides a standard interface to plug in a range of multi-resolution algorithms for measurements. The testbed sidesteps the tedious job of building real-time rendering benchmarks and makes consistent performance measurements and comparison possible.

- Using the framework, three level of detail (LOD) algorithms are measured in several real-time rendering benchmarks. The measurement results are used to effectively compare their performance.

1.4 Outline of material

We begin with the motivations behind this research. Then, details on multi-resolution algorithms for real-time rendering and their performance issues are discussed in chapter 3. The considerations in performance evaluation in general and related performance evaluation work are also discussed in this chapter. Having established this background information, we present the performance evaluation framework in chapter 4. The key performance measures in the framework are presented in chapter 5. Chapter 6 introduces the design of the real-time rendering testbed. Chapter 7 presents a prototype system and the methods to plug in algorithms and conduct performance measurements with it. Chapter 9 presents the performance evaluation and comparison of three LOD algorithms using our framework. The last chapter summarizes the research and presents topics for future research. It concludes the dissertation. To highlight certain design choices and techniques, Appendix A is included. It contains details on the prototype implementation of the performance evaluation framework.

Chapter 2

Motivation

2.1 Observations

The initial drive for building a performance evaluation framework of multi-resolution algorithms for real-time rendering came from our virtual reality research. We have developed several virtual reality packages over the past decade [51]. The most famous ones include the MR Toolkit and MR Objects. They facilitate the low level support of virtual reality applications. However, scene content has to be kept as simple as possible to maintain the required frame rate. In order to add rich interesting scene content to our applications and maintain the required frame rate, we started to look at current real-time rendering algorithms and try to incorporate them into our virtual reality packages.

Among a variety of real-time rendering approaches, multi-resolution algorithms have drawn increasing attention recently. The fundamental idea of multi-resolution algorithms is to describe 3D models and their attributes such as color and texture in a variety of resolutions. Depending on the time budget, object's distance and other factors, the appropriate level of detail within the model is chosen for rendering. Thus, the required frame rate is maintained and the "best" image of 3D scenes is displayed. Dozens of algorithms have been proposed for the real-time rendering problems. Quite often the papers presenting the algorithms do not provide a thorough evaluation of their performance. Most of them only concentrate on presenting the technical strategies, such as how a model hierarchy is built and how the simplification methods are novel and unique. Little performance evaluation is reported. There is no comparison between the algorithms either. It is very difficult to determine how well these algorithms perform in real-time rendering applications.

To find which algorithms perform better in real-time rendering applications, fair measurements must be done. We first experimented with Isler's real-time multi-resolution algorithm[31] to see how it performs for real-time rendering [67]. This algorithm was developed to adaptively change the resolution of a triangular model during real-time rendering. The following were observed in the performance experiment:

- The algorithm is capable of switching level of detail back and forth and increasing the frame rate. It also introduces artifacts. However, these characteristics are hard to describe and compare with other algorithms. This is because there has been no performance metrics and automatic measures defined for multi-resolution algorithms.

- A real-time rendering task is complex, time consuming and tedious to build. It is not only concerned with modeling a 3D scene and arranging a navigation path, but also with many hardware and support software configurations. It also results in numerous task parameters that need to be controlled during the evaluation.

- The algorithm performs differently on different tasks. For example, for a simple scene with 4k polygons, the preprocessing time is 30 seconds. However, for a large model with 69k polygon, the preprocessing time is more than one hour on the same platform. It suggests many different real-time rendering tasks are needed to obtain fair measurement results.

- In this experiment, the algorithm implementation was bundled with the real-time rendering tasks in the program, which makes it difficult to reuse the task for other algorithms without changes. A clean and well defined interface for real-time rendering tasks is needed for plugging in different algorithms.

When we looked back into the field of real-time rendering, the following facts were revealed more clearly:

- There is no previous work on conducting broad measurement and comparison of multi-resolution algorithm or any other real-time rendering algorithms.

- Like other performance evaluation problems, performance evaluation of multi-resolution algorithm for real-time rendering needs to be systematically studied.

2.2 Problem definition

The above observations lead me to try to find a solution for performance evaluation of multi-resolution algorithms, which is a performance evaluation framework. The framework consists a set of well defined performance measures and a real-time rendering testbed which can load and run various real-time rendering benchmarks. Multi-resolution algorithms can be automatically measured and compared in this standard environment, producing consistent and meaningful performance results and comparisons.

The key research problems in the performance evaluation of real-time rendering algorithms addressed in thesis thesis include:

- **Performance metrics :** Many multi-resolution algorithms have been proposed for real-time rendering in the literature. However, the main performance targets and typical artifacts that impair the performance of algorithms have not been clearly defined. Identifying the performance metrics is the first problem that needs to be solved in this research.

- **Automatic measures :** Automatically measuring and comparing multi-resolution algorithms is one of the research goals. The automatic measures of each of the metrics is required to achieve this goal. They are expected to detect typical artifacts and physical properties of the algorithms, and be computable and easy to interpret.

- **Real-time rendering benchmarks:** The benchmarks are typical real-time rendering tasks which are used to evaluate algorithms. They are essential for measuring the algorithms. A real-time rendering task tends to be complex, tedious and difficult to build and run. It usually has many parameters and attributes, such as viewer positions, orientations and behavior, virtual environment lighting, object hierarchy, object graphic attributes and behavior. Also, many different hardware and software configurations need to be involved, making the job of composing such a task even more difficult and time consuming. A number of tasks of various types are needed. Thus, instead of one or two real-time rendering benchmarks, proper techniques to compose, load, and run many typical real-time rendering tasks must be investigated and developed. They should relieve the tedious job of building various real-time rendering benchmarks, and also provide a standard interface for various multi-resolution algorithms to be plugged in and tested.

The thesis research is to investigate the above problems and propose solutions. The objective of this thesis is to propose a performance evaluation framework of multi-resolution algorithms for real-time rendering. It consists of a set of automatic performance measures and a standard real-time rendering testbed. This framework can automatically measure a range of multi-resolution algorithms while they perform various real-time rendering benchmarks. The performance results are consistent and can form the basis for performance comparison.

2.3 Chapter summary

The motivation behind this research came from the need to choose optimal multi-resolution algorithms for real-time rendering application. This need was further identified as building a performance evaluation framework.

Various multi-resolution algorithms have been proposed. However their performance has not been evaluated and compared. A standard evaluation framework which consists of a set of objective measures and a testbed will help researchers to measure and compare existing and new multi-resolution algorithm. This research attempts to build such a framework.

Chapter 3

Background and Related Work

This chapter provides an overview of background material used throughout the rest of this dissertation. First real-time rendering is briefly reviewed, as it is the application domain of multi-resolution algorithms. Then, we will discuss typical multi-resolution algorithms and examine the need for a performance evaluation framework for evaluating and comparing their performance. Lastly, the general guidelines of computer system performance evaluation and some related performance evaluation work are discussed.

3.1 Real-time rendering

In the area of computer graphics, real-time rendering is concerned with displaying 3D scenes rapidly on the computer. Given a set of 3D objects, their attributes, lighting and a virtual camera, a 2D image is rendered on a screen. The viewer can interact with it, and the feedback affects what is displayed next. The cycle of reaction and rendering happens at a rapid enough rate that viewers see an animated environment and feel immersed in it. This is the core of the real-time rendering procedure.

The underlying tool for supporting this procedure is the rendering pipeline. Figure 3.1 shows the conceptual view of the pipeline.

The pipeline consists three major stages, application, geometry and rasterization. The slowest stage determines the rendering speed. Thus, the goal of real-time rendering is to optimize the whole rendering pipeline, so that complex and large scale 3D virtual environments can be rendered interactively. Many software and hardware techniques have been developed to optimize each stage and increase the performance

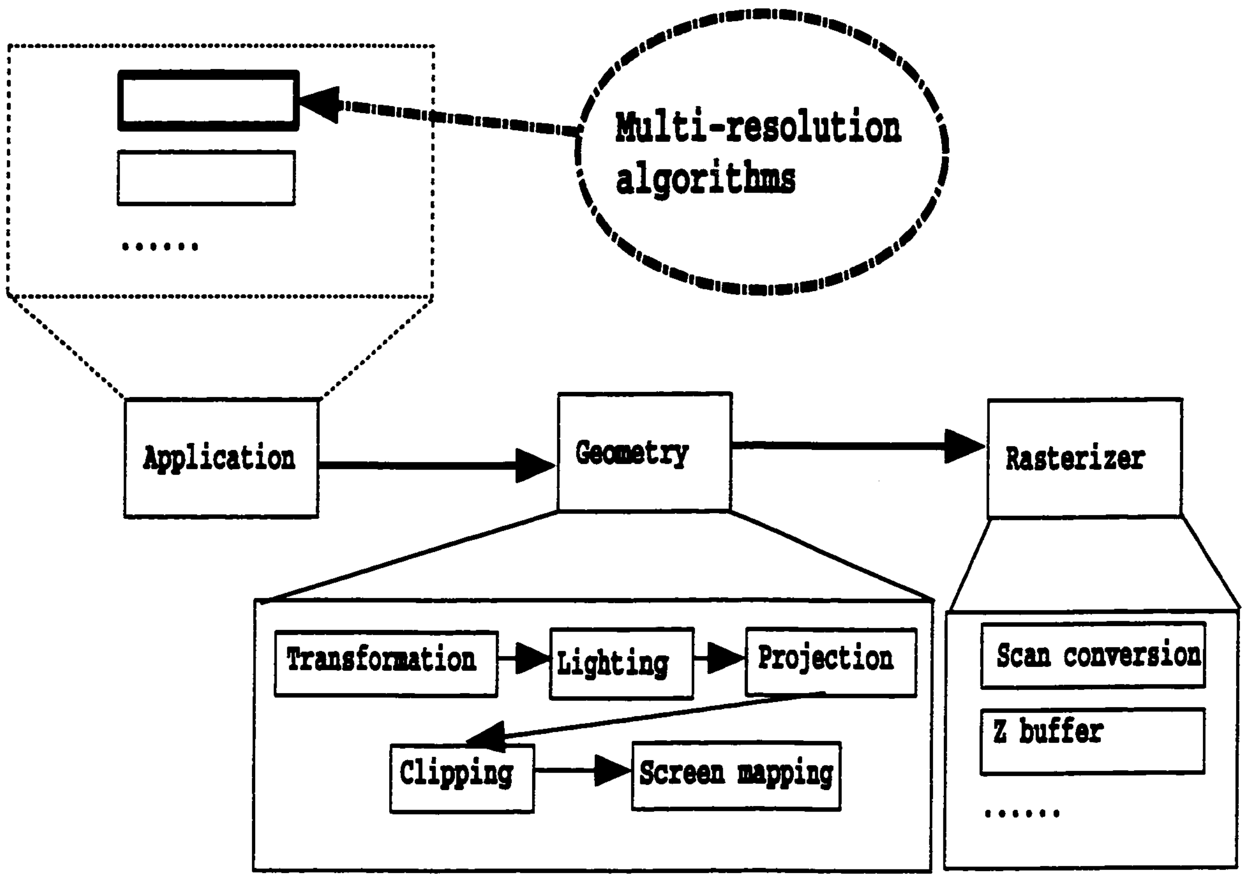


Figure 3.1: Conceptual view of the rendering pipeline

of the pipeline. Moller and Haines's real-time rendering book provides a good reference for these techniques [43].

Multi-resolution algorithms are one type of real-time rendering technique. They try to optimize the rendering pipeline and solve the real-time rendering problem in the application stage. For any given system, available hardware capacity, such as primitive drawing rate, frame buffer fill rates, transformation and lighting throughput, and network bandwidth is essentially fixed. But the complexity of the scene may vary considerably. Sometimes, many details only waste rendering time and do not contribute, or contribute little, to the quality of the displayed images. Multi-resolution algorithms try to choose proper approximations to maintain an interactive and constant frame rate.

3.2 Multi-resolution algorithms survey

A multi-resolution model is a model representation which captures a wide range of approximations of an object and which can be used to reconstruct any one of these on demand [23]. The multi-resolution modeling approach relies on these representations of a virtual environment. When the time budget is tight, a coarser representation of the objects is rendered. In this way, the target frame rate is guaranteed and the "best" possible image is produced. Objects' size, distance from the view point, objects' inherent importance in the scenario and other factors can also be used as threshold parameters for selecting and rendering the approximations. Multi-resolution models and the mechanism for selecting and rendering the approximations together constitute a multi-resolution algorithm. Multi-resolution modeling methods can be classified as discrete multi-resolution, which is also called level of detail modeling, and continuous multi-resolution modeling.

3.2.1 Level of detail

Level of detail modeling consists of a set of pre-generated increasingly simpler models. These models are usually generated off-line. During real-time rendering, a renderer selects one level of detail model to use and renders that model in a given frame according to some threshold. One of the typical level of detail modeling approaches is described by Funkhouser and Sequin [21]. They generate the levels of detail representations of a 3D building model manually and use a time management strategy, i.e. frame rate, to select the level in their real-time walkthrough system. Their algorithm yields good results on maintaining consistent frame rate. However image quality is not evaluated and analyzed adequately in their work. Support for levels of detail has also been included in a number of commercial rendering systems, including RenderMan [54], Open Inventor [61], IRIS Performer [46], and Cosmo Worlds [64]. The RenderMan interface provides for mixing successive levels of detail together, but leaves the exact mechanism undefined. Performer provides explicit support for smooth transition between level of details, such as alpha blending and geomorphing (see detail in §3.2.3). Cosmo Worlds is capable of creating alternative representations of an object with varying levels of detail and displaying a different version based on distance or frame rate. Most current level of detail techniques mainly address the problem of level of detail control in real-time rendering. Generation of level of detail models are done either manually or with mesh simplification algorithms (see §3.2.4).

Level of detail modeling is simple and easy to use. However it is not flexible for real-time rendering applications. The levels of detail available at run time are limited. Usually, there are three or four levels for one object due to resource limitations. A renderer would be forced to pick one of them, even if it needed an intermediate level. Thus, the renderer would either have to pick a model without sufficient detail (and sacrifice image quality) or choose a model with excess detail (and waste time). Level of detail modeling approaches usually suffer from popping artifacts and abrupt scene changes caused by switching the level of details. Other artifacts that level of detail modeling may produce include loss of geometric, topological, luminance and color information, non-guaranteed or fluctuating frame rate, lengthy preprocessing procedures, and extra resources for storing the huge amount of data. Performance evaluation is to detect these artifacts.

3.2.2 Continuous multi-resolution

As we have seen level of detail modeling techniques have drawbacks for real-time rendering, multi-resolution modeling approaches which can continuously adapt the representation at run time based on performance or viewing conditions are certainly in need. These techniques are called continuous multi-resolution modeling.

Continuous multi-resolution modeling techniques have been in use in height field applications, such as terrain [15] [39]. Most are based on a regular subdivision (e.g., quadtrees) of the height field surface. Recently, many continuous multi-resolution modeling approaches have been proposed for general polygonal surfaces. Multi-triangulation [18] [17] is a general framework for describing multi-resolution of triangulated surfaces. Vertex hierarchies have drawn increasing attention recently. They focus on generating surface hierarchies and related operations which can facilitate varying the level of detail over different parts of the model. One of the typical vertex hierarchy algorithms is Hoppe's *progressive mesh* [29] [30]. It is a scheme of storing an arbitrary mesh as a much coarser mesh together with a sequence of n detail records that indicate how to incrementally refine the base mesh exactly back into the original mesh. This scheme can be used to adaptively transmit and render large geometry models. However the performance of the algorithm has not been evaluated against other algorithms. Garland introduced a vertex hierarchy which is similar to progressive mesh. He proposes to build the vertex hierarchy using his quadric based simplification algorithm [23]. He did a good performance analysis on his surface simplification algorithm. But more experiments are needed to test how the vertex hierarchy based on his simplification algorithm behaves in real-time rendering. Luebke and Erikson's idea [41] is to create a vertex tree using a chosen simplification algorithm during a preprocessing stage. This vertex tree can then be used to selectively vary the level of detail according to changing view parameters. No performance data except for preprocessing time is provided in their paper. Xia and Varshney [65] introduce a merge tree for a triangular mesh, and then use it to guide the selection of appropriate triangles for display. They provide some convincing performance results in their paper. The algorithm still needs to be compared with other algorithms. Lau *et. al.* present a simplification list [34], which is similar to Hoppe's progressive mesh. It decimates an original mesh using edge collapse strategies and keeps the simplification procedure in memory, which is later used to reduce or increase level of detail adaptively in run time. The performance and image quality also need to be evaluated thoroughly.

The above multi-resolution modeling approaches all need long preprocessing time and the generated data structure may consume a fair amount of memory. Therefore, the continuous multi-resolution models are usually generated off-line, with the exception of Lau's simplification list, and stored in a repository for later rendering. These multi-resolution modeling approaches are termed as *static continuous multi-resolution modeling*. The other type of continuous multi-resolution modeling algorithms concentrate on time management and adapting the level of detail continuously over successive frames during real-time rendering. They are defined as *dynamic continuous multi-resolution modeling*. There has been comparatively less work on dynamic continuous multi-resolution modeling for real-time rendering. Green [25] introduces geometry compilation for large scale virtual reality applications. His idea is to use a geometry compiler to generate a proper tessellation of n geometric object at run time. It avoids the preprocessing work of generating multi-resolution models and it will maintain a certain frame rate and preserve the geometric and topological information of objects more easily. It is suited for predefined geometries, such as cone, sphere, and torus, but leaves arbitrary meshes with no definition. The performance of this method also needs to be investigated.

3.2.3 Smooth transition between multi-resolution models

Both level of detail modeling and continuous multi-resolution modeling can cause visual artifacts in real-time rendering. The number of polygons and appearance in two models can be significantly different. If the switch of these two models occurs in consecutive frames, users will experience popping artifacts and abrupt scene changes. Funkhouser *et. al.*[21] propose to use *alpha blending* to smooth the transition between the two models. Visual artifacts are reduced. However, the rendering cost increases significantly because the system must render two levels of the model at the same time. Another alternative is *geomorph* [28]. The idea is to smoothly interpolate between the geometries of two consecutive levels over several frames. Suppose that we are transitioning between a model M and a simpler model $M0$. For each vertex v in M, we substitute an interpolated position $tv + (1-t)\phi(v)$. At $t = 0$, the model will have exactly the same shape as M, and at $t = 1$, the model will have the shape of $M0$. By moving t between 0 and 1 over several successive frames, we can smoothly transition between the two models. However, there is the additional overhead of interpolating the vertex positions for each frame. Experiments are needed to see how expensive geomorph is and if it is faster than just rendering the original model.

3.2.4 Mesh simplification algorithms

As discussed in previous sections, one of the fundamental support tools for multi-resolution modeling is surface simplification algorithms. In practice, most of the multi-resolution models, both level of detail and continuous multi-resolution modeling are created by simplification algorithms. To have a better understanding of multi-resolution modeling, a brief review of the most relevant simplification algorithms is presented. Cignoni *et. al.* [7] and Heckbert and Garland [27] have more detailed and complete surveys of simplification algorithms.

Surface simplification aims at reducing the number of polygons while assuring a good approximation of the original model. Typical algorithms include vertex decimation, vertex clustering, iterative edge collapse and vertex pair contraction. Vertex decimation methods iteratively select a vertex for removal, remove all adjacent faces, and retriangulate the resulting hole, e.g. [49] [6]. Vertex clustering methods [47] divide the original models into a grid. Within each cell, the vertices are clustered together into a single vertex, and the model faces are updated accordingly. Edge collapse algorithms are drawing increasingly attention among various mesh simplification methods recently. They iteratively collapse edges, e.g.[28] [29] [34]. The essential difference between these algorithms is to choose which edge to collapse and how to collapse. Vertex pair contraction methods extend the contraction pairs to non-edges to facilitate better approximation, e.g. [22] [40].

Other simplification algorithms not only take into account surface attributes but also color, material and textures. Maciel and Shirley [42] simplify a scene using a mix of approaches, including geometric simplifications created by Iris Performer, texture maps and colored bounding boxes. Chamberlain et.al [5] construct a spatial hierarchy of cells over the scene and associate with each cell a color box. Cohen et.al [9] introduce a simplification algorithm which preserves appearance. They decouple a surface and its attributes and then apply color and normal map after the simplification. Their algorithm can preserve the appearance, but is relatively expensive.

All of simplification algorithms approximate the original object using a coarser representation. However, it is not clear if the simplification procedure and the rendering of the simplified representation is less expensive than rendering of the original object. The image fidelity generated by these algorithms has also never been thoroughly evaluated and compared.

3.2.5 Geometric measures of simplified meshes

Research has been done on intrinsic geometric measures between the original mesh and the simplified mesh. Cignoni et al. propose a tool, Metro [8], to compare the geometric differences between a pair of surfaces (e.g. a triangulated mesh and its simplified representation). It adopts a surface sampling approach and uses surface distances as the major error measure. In their work [8], they provide some experimental results of some mesh simplification algorithms produced by Metro. However, the parameters of these experiments are not provided, and the results are not explained and analyzed in the report. Nevertheless, this is the first report about the geometric measurement and comparison of mesh simplification algorithms.

Geometric measures of the simplified mesh is for estimating the similarity of geometric shape. In real-time rendering, one would rather like to know the similarity of appearance of multi-resolution models. A geometrically distorted surface model does not necessarily produce a distorted appearance in real-time rendering. When the distorted portion is not visible, the appearance is not affected. Therefore, geometric measurements only play roles in the middle stages. Even if the measures were fully developed, one would still need image fidelity measures of multi-resolution algorithms in real-time rendering.

3.2.6 Multi-resolution algorithms variants

View-dependent multi-resolution algorithms

Visibility Culling algorithms accelerate rendering by avoiding the rendering of objects that are not visible in the image. The idea is classic and has been used in computer graphics for more than two decades. Recently, algorithms have been proposed to combine multi-resolution techniques with visibility culling for real-time rendering. [41, 29, 30, 65]. They are often referred to as view-dependent multi-resolution. The idea is to model a surface in a hierarchical fashion. The simplification process continuously queries this hierarchy to generate a scene containing only those polygons that are important from the current viewpoint. Xia et.al [65] first explores this idea. Luebke et.al [41] develop a general framework for dynamic view-dependent simplification. Hoppe [29, 30] extends the idea to the progressive mesh.

In real-time rendering one major performance shortcoming for view-dependent multi-resolution algorithms is fluctuating and non-guaranteed frame rate. This is because the performance of culling algorithms depends on the scene complexity and depth. Scene complexity and depth in a large virtual environment may vary from one part to another. Therefore, for some viewpoints, culling computation can be much more expensive than others. Second, for some viewpoints, most of the scene is visible, and culling does not help to increase rendering speed no matter how good it is. Importantly, querying the hierarchy costs time. If this process uses up the time saved for rendering, these algorithms do not improve rendering speed. Other artifacts are long preprocessing procedures to construct an object hierarchy or scene hierarchy of the virtual environment. These algorithms may cause loss of geometry and color information due to inaccurate depth computation and simplification. However, popping artifacts and abrupt scene changes usually are not major issues.

Hierarchical image cache

The Hierarchical Image Cache method is another variant of multi-resolution algorithms. The idea is to hierarchically model a 3D scene and store it in image caches. During the interactive rendering period, instead of rendering all the original geometric models, the cached images of far away geometries are reused as textures mapped to the polygons which represent its boundary. This idea was first investigated by Regan and Pose [45]. It is called Multiple Frame Buffer Rendering. It relies on hardware with multiple frame buffers. Objects are organized into different buffers based on their distance from the viewer. These frame buffers are updated at different rates. The frame buffers which contain close objects are updated frequently, while distant parts are updated at a much slower rate. The reason is that distant objects do not change or move as much as close ones. They do not need to be updated in every frame. Thus only a small portion of the environment needs to be updated frequently. The final image is created by overlapping the frame buffers form front to back. Recently, some software solutions for hierarchical image cache, have been proposed [42] [50] [48] [2]. They put more effort on the hierarchical organization of the scene, preservation of the image quality or time management. However, the performance measurements of this approach still need to be done.

3.2.7 Section summary

In this section, some typical multi-resolution algorithms and their variants were presented. It is shown that all these algorithms try to solve the real-time rendering problem by trading off rendering quality for real-time. However, it is not clear how they perform in real-time rendering applications and which ones perform better than others in terms of frame rate, image fidelity and other performance considerations. Thorough measurement and comparison of these algorithms are needed to answer this question. In the mean time, we can also see that there are considerable differences between algorithms. To do a fair measurement and compare them, a standard performance evaluation framework is required. It is the beginning and foundation for the performance study of the algorithms.

3.3 The art of computer system performance evaluation

Performance evaluation seldom stands alone as a research topic. It is always coupled with some application area - science (e.g. [12]), sociology (e.g. [20]), engineering (e.g. [71]), or anywhere there is a performance concern. Performance evaluation of multi-resolution algorithms is a problem in the category of computer system performance evaluation (A computer system is referred to any collection of hardware, software, and firmware components). A great amount of research work has been done in this category, such as performance evaluations of computer hardware systems, databases, networks, and various algorithms. Most of these problems, including the performance of multi-resolution algorithms itself, are unique. The evaluation techniques, measures, and benchmarks used for one problem generally can not be used for the next problem. However, general methodologies and systematic approaches have been studied to help performance analysts to solve day-to-day performance measurement problems and obtain the most performance information with the least effort [32] [36]. In this section, we review these general guidelines and emphasize the key components for our performance evaluation of multi-resolution algorithms.

3.3.1 Goals

The first step in any performance evaluation research is to state the goals. The goals define the performance evaluation system boundaries. Given the multi-resolution algorithms, for example, the goal may be to measure if and how much the algorithms improve response time and image quality of a hockey video game on the $SonyPlayStations^{TM}$. In this case, the system would consist of not only the algorithms, but also the particular rendering engine of the game and the hardware support platform. Thus, the study results may significantly depend on many components other than the algorithms. On the other hand, if the rendering mechanisms and graphics workstations are similar except for the difference between the algorithms and the goal is to decide which algorithm is better for general real-time rendering tasks, the algorithms may be considered the one and only measured target. Other components are parts of the support platform.

The goals also determine how to develop a performance evaluation system and how to proceed with the performance evaluation study. They affect the architecture of a performance evaluation system, and the performance measures as well as benchmarks used to compare the measured algorithms.

In a previous section, the state of art of multi-resolution modeling were described. Many algorithms have been proposed, and new algorithms are being and will be presented. However, their performance has never been evaluated. Thus, our goals are to develop the first performance evaluation framework. It will be used to automatically measure and compare multi-resolution algorithms in a standard environment. The framework is easy to extend with new measures as new multi-resolution techniques are developed.

3.3.2 Evaluation techniques

The three broad techniques for performance evaluation are measurement, simulation and analytical modeling. Different techniques are chosen based on different conditions. Measurement is usually used in the post-prototype stage, where the evaluated systems are available, evaluation tools are available and the requirements of time and level of accuracy are moderate. Simulation and analytical modeling are used when systems are not available, theoretical modeling or simulation techniques are mature, and the accuracy requirement is moderate. In general, measurement tends to be more expensive than simulation and analytical modeling. Its results are also more convincing to users than the other two techniques.

The measurement methods can be classified into objective measurement and subjective measurement. The objective methods are repeatable, quantifiable and can be used in real time. They are usually implemented in software. Subjective assessment is time-consuming and the results are difficult to duplicate. However, the beauty of subjective assessment is that it measures the perceived quality directly.

3.3.3 Performance measures

Appropriate performance criteria or metrics are the basis for meaningful performance evaluation. For any performance study, a set of performance metrics must be chosen. They should be thorough, specific, measurable, and realizable. Listing the services offered by the measured algorithms is a good way to choose performance metrics [32]. Multi-resolution algorithms are proposed for real-time rendering applications. Ideally, the algorithms should increase and maintain the frame rate at a user-specified value while preserving image quality as much as possible. Therefore, increasing and maintaining a certain frame rate and preserving image fidelity are the key performance requirements. In addition, the algorithms shouldn't require huge amounts of extra data which could be several times larger than the original model. This not only increases the workload of generating the data, but also involves memory management problems if the data set is too large. The model preprocessing work is also an overhead for a good algorithm. Extra hardware requirements, such as multiprocessors is also a restriction for general purpose multi-resolution algorithms. When defining the performance metrics, all the requirements of frame rate, image fidelity, resource consumption, and preprocessing time should be taken into consideration.

Performance measures are the methods of observing and quantifying if and how well the test subjects satisfy the performance criteria when they perform tasks. They are also called performance monitors in the performance evaluation literature. After the performance metrics are chosen, the major concern in objective performance evaluations is to design the performance measures. The detail of our performance measures is described in a later chapter.

3.3.4 Benchmarks

In the performance evaluation literature, the process of performance comparison of two or more systems by measurements is called benchmarking, and the workload used in the measurements are called benchmarks [32]. In the context of our performance evaluation of multi-resolution algorithms, this tradition is followed. *Benchmarks* are the real-time rendering tasks used in the performance measurements.

Benchmarks are the most crucial part of any performance evaluation project. They must be well defined in order to accomplish the performance evaluation. In general, two major aspects need to be considered when selecting benchmarks.

- Completeness: view the test subjects as a service provider. Benchmarks should exercise the provided services as completely as possible.

- Representativeness: A benchmark should be representative of the real application. It should represent the latest usage pattern of the test subject. In the mean time, it should also be kept minimal and self-complete.

Our real-time rendering benchmarks are designed based on the above general guidelines. They also have their own characteristics.

3.3.5 Analysis and comparison

Performance evaluation system design is often the major concern of performance evaluation research. However comparing two or more test subjects by analyzing the generated performance data is also a common problem. This problem is concerned with using proper statistical techniques to analyze the samples and compare several alternatives. Data analysis and comparison has been a classic problem in both statistics and performance evaluation, and has been studied for many years [32] [36]. Various data analysis models and methods are presented in the literature. However, we are not concerned with complex models and analysis in this performance evaluation study. Multi-resolution algorithms and real-time rendering are still new research areas. This performance study is considered a first step in the systematic performance analysis and comparison of multi-resolution algorithms. Simple analysis is preferred because it is easy for other researchers to understand and can expedite the improvement of real-time rendering algorithms.

Our research focus is on providing a standard performance experiment environment, so that various experimental measurements can be designed. Complex and in depth data analysis and performance comparison will be left to the multi-resolution algorithms researcher. The performance analysis and comparison of some typical algorithms only serves as the evaluation of our framework itself.

3.3.6 Section summary

The general considerations of computer system performance evaluation have been reviewed in this section. Jain [32] and Lavenberg [36] have presented detailed descriptions of the methodologies. Their books are good references for any computer system performance evaluation research. Our performance evaluation problem is a specific case in this domain. The experiences in computer system performance evaluation can be employed as general guidelines. However, performance evaluations of multi-resolution algorithms have unique characteristics, especially in the performance measures, real-time rendering benchmarks, and system design. Investigating these issues and working out solutions are the major thrusts in this research.

3.4 Related performance evaluation work

The background of real-time rendering, multi-resolution algorithms and computer system performance evaluation has been discussed in previous sections. Performance evaluation of multi-resolution algorithms for real-time rendering is seen as the first research attempt in the intersection of these areas. However, there is some related work to our problems in the areas of graphics workstations performance evaluation, performance evaluation of digital video compression techniques and geometric measurement of simplified meshes. The following sections describe how these works are related to our problem and also how they differ from the performance evaluation of multi-resolution algorithms.

3.4.1 Graphics workstations performance evaluation

Work has been done on graphics workstation performance measurement. In the 1980's, the Graphics Performance Characterization (GPC) group of the National Computer Graphics Association (NCGA) proposed four levels of performance measurement for graphics workstations. These levels are low-level primitives(points, lines and polygons per second), pictures, systems(input and action response times), and applications. Many attempts have been made at measuring the four levels of graphics workstations performance [71] [53]. But, to our best knowledge, no fully functional performance evaluation system has been designed and implemented so far. The graphics workstation performance work concentrates on measuring graphics hardware performance over a range of graphics tasks. These tasks include 2D graphics, 3D graphics, windows, visualization tasks and many more. This makes graphics workstation performance measurement a large and complex problem.

The OpenGL performance characterization project [44] is an alternative to graphics workstations performance evaluation. It was originated by an ad-hoc project group, OpenGL Performance Characterization (OPC) in 1993. It is aimed at providing unambiguous methods for comparing the performance of OpenGL implementations across vendor platforms, operating systems, and windowing environments. Two benchmarks have been released by OPC. The first one, Viewperf, is for measuring the 3D rendering performance of systems running under OpenGL. It takes a data set and rendering parameters from command line arguments and outputs the frame rate of the test. Frame rate is the only performance metric. The second one, GLperf, which complements Viewperf, is for measuring the optimal performance of 2D and 3D graphics primitives across vendor platforms. The output includes number of polygons per second. OPC also selected some benchmarks representative of the OpenGL rendering portion of Independent Software Vendor (ISV) applications called viewsets to satisfy the real-world benchmarking requirement. A viewset is a group of individual runs of Viewperf that attempt to characterize the graphics rendering portion of an ISV's application.

The OpenGL performance characterization is an on going development project. It is currently focused on measuring the speed of graphics systems under the OpenGL API. Meaningful performance metrics except frame rate still need to be defined.

Our work is different from the work on graphics workstation performance evaluation and the OpenGL performance characterization project. It is measuring multi-resolution algorithm performance for a particular task, which is real-time rendering for interactive 3D graphics. This is a single, well defined task, thus removing the problem of establishing a set of measures that cover a diverse set of graphics tasks. The effects of different graphics platforms can be reduced by evaluating the algorithms on the same platform. Also, when measuring the performance of the algorithms, many aspects need to be considered, such as frame rate, static image quality, temporal image quality etc., in contrast to OpenGL performance characterization project, which is mainly concerned with the speed of graphics workstations.

3.4.2 Performance evaluation of digital video compression techniques for high definition TV (HDTV)

This problem is concerned with measuring how well the video compression techniques preserve the level of quality of the original video with as few bits as possible. A fair amount of work has been done in this area. They can be classified into:

- Modeling and General Discussion. In [70], Zou surveyed various lossless and lossy compression techniques and discussed the artifacts that these techniques bring in, and the general approaches of objective and subjective evaluations of these techniques. Lambrecht [56], according to a study of the human visual system, developed a spatial-temporal model for designing the image quality metrics of coded video from the visual perception point of view.

- Subjective Measurement. Lauzon [35] conducted a formal subjective assessment of MPEG-2 video coding technique on selected HDTV sequences. Bit rate, structure of picture organization and temporal processing are the performance metrics they used for testing. Dezhgosha [14] analyzed their subjective test results on VQ based image coding algorithm. He also did objective measurements using NTSC video performance metrics. But the meaning and validation of the metrics were not discussed.

- Objective Measurement. Wolf proposed a performance evaluation system [62] for objective quality assessment of digitally transmitted video. She described some spatial features and temporal features that need to be measured objectively, but it is not clear how these features are quantified in her system. Also, no detailed experimental methods and results are reported. Lambrecht [56] [55] developed a Spatial-Temporal Model of HVS for assessing MPEG coding fidelity. He estimated the free parameters of the model based on psychophysics experiments on coded video sequences. The parameterized model is then used as the basis for a coded video sequences quality metric. Since the parameters of this model are particular for MPEG coded video sequences, it is not clear how useful it is for measuring multi-resolution algorithms.

In the literature of performance evaluation of digital video compression, such as [70] [69] [62] [58] [35] [14], the results on subjective measurement are found to be fruitful. But few methods or systems are reported to conduct objective measurement, of either absolute distortion or visible error between the referenced image sequence and the tested image sequence. However, the performance evaluation of digital video compression techniques is a valuable reference for performance evaluations of multi-resolution algorithms for its characteristic of measuring the degradation of image sequence quality.

Performance evaluation of digital video compression techniques is similar to our problem, in the sense that both need to measure the image degradation caused by techniques or algorithms. However, the nature of the two problems are different:

- *Goal:* As we discuss here, video compression techniques are concerned with representing video images with as few bits as possible without losing or discarding important information. Bit rate is a big performance concern for measuring video compression techniques. Multi-resolution algorithms are concerned with increasing frame rate while preserving required image quality. Frame rate is certainly a major performance metric.

- *Artifacts impairing image quality:* The artifacts they cause are different. Digital video compression techniques may produce quantization noise, loss of contrast, loss of resolution, loss of chrominance, edge busyness, motion jerkiness and ringing. These artifacts are usually seen over the whole image. Multi-resolution algorithms, however, mainly cause lose of geometric and luminance information statically, and popping effects, sudden image changes and fluctuating frame rate temporally. These artifacts are usually seen locally. For example, when the level of detail of an object changes, only the image in the neighborhood of that object is effected, the artifact doesn't spread to other parts of the image.

- *Comparison of image sequence fidelity:* Digitally compressed video can always be compared with the original video, statically or temporally to determine its quality. That is, there is a well defined standard for its performance. This is rarely the case with real-time rendering. There is no reference algorithm that can display the environment at the optimal display rate with perfect image quality, otherwise there would be no need for multi-resolution algorithms and other real-time rendering techniques. This difference stems from the fact that video compression is used to decrease bandwidth and storage requirements, and doesn't need to decrease display time.

3.5 Chapter summary

From this brief survey of relevant work, we can make the following observations:

First, there is a need for a performance evaluation framework of multi-resolution algorithms, so that various algorithms can be measured in a standard environment, generating a relative ranking of their performance.

Second, the major research problems in this study are the performance measures and real-time rendering benchmarks.

The next chapter presents our performance evaluation framework for multi-resolution algorithms that can cope with these problems.

Chapter 4

Overview of the Framework

In the previous chapters, we discussed the motivation for this research and reviewed various relevant work. Now, we will present the performance evaluation framework of multi-resolution algorithms for real-time rendering that has been developed. This chapter focuses on the design principles and fundamental ideas of the framework. The key components of the framework — automatic performance measures and the real-time rendering testbed will be described in much greater detail in chapters 5 and 6. The prototype system and experimental results are described in chapters 7 and 8. Lastly, the applications of the framework are discussed in chapter 9.

4.1 Design principles

As we have seen in the discussion of related work, many multi-resolution algorithms have been proposed, however, there have been no performance measurements and comparison of them. There are two main reasons. First, the artifacts of multi-resolution algorithms have not been fully recognized. There have been no performance metrics and automatic measures in real-time rendering research. Second, automatic performance evaluation of multi-resolution algorithms involves many complex hardware and software configurations. It is tedious and time consuming to build real-time rendering benchmarks. The primary goal of this research is to overcome these problems. We intend to design a standard performance evaluation environment, which consists of a set of well defined performance measures and is able to load various real-time rendering benchmarks. It is aimed at measuring existing multi-resolution algorithms of various types in a standard real-time rendering environment and automatically producing their performance data.

Our work is based on two assumptions.

- The performance evaluation framework is platform independent and can be ported to test multi-resolution algorithms across different hardware and software platforms. However, it is used to measure and compare the algorithms on the same hardware and software platform, to guarantee the consistency of the measurement results.
- No single performance measure. Algorithms are tested over various real-time rendering benchmarks to avoid biased measurement.

As a performance evaluation framework, it is by no means complete. We expect it to be extended to add new measures and incorporate broader real-time rendering benchmarks in the future.

4.2 System architecture

The fundamental idea of the framework is that algorithms are evaluated and compared in a standard real-time rendering environment. The framework contains a number of automatic performance measures and provides a standard testbed which can load and run various real-time rendering benchmarks. Multi-resolution algorithms can be attached to the framework via a standard software interface. The performance is measured automatically in the system while an algorithm performs real-time rendering benchmarks. The performance data can be used to generate the relative rankings of algorithms. Figure 4.1 shows the logic diagram of this framework.

The two fundamental components of the framework are: performance measures and real-time rendering testbed.

4.3 Performance metrics and measures

The term performance metrics refers to the criteria used to evaluate performance. For example frame rate - frames per second could be used to compare two multi-resolution algorithms. For the performance study of multi-resolution algorithms,

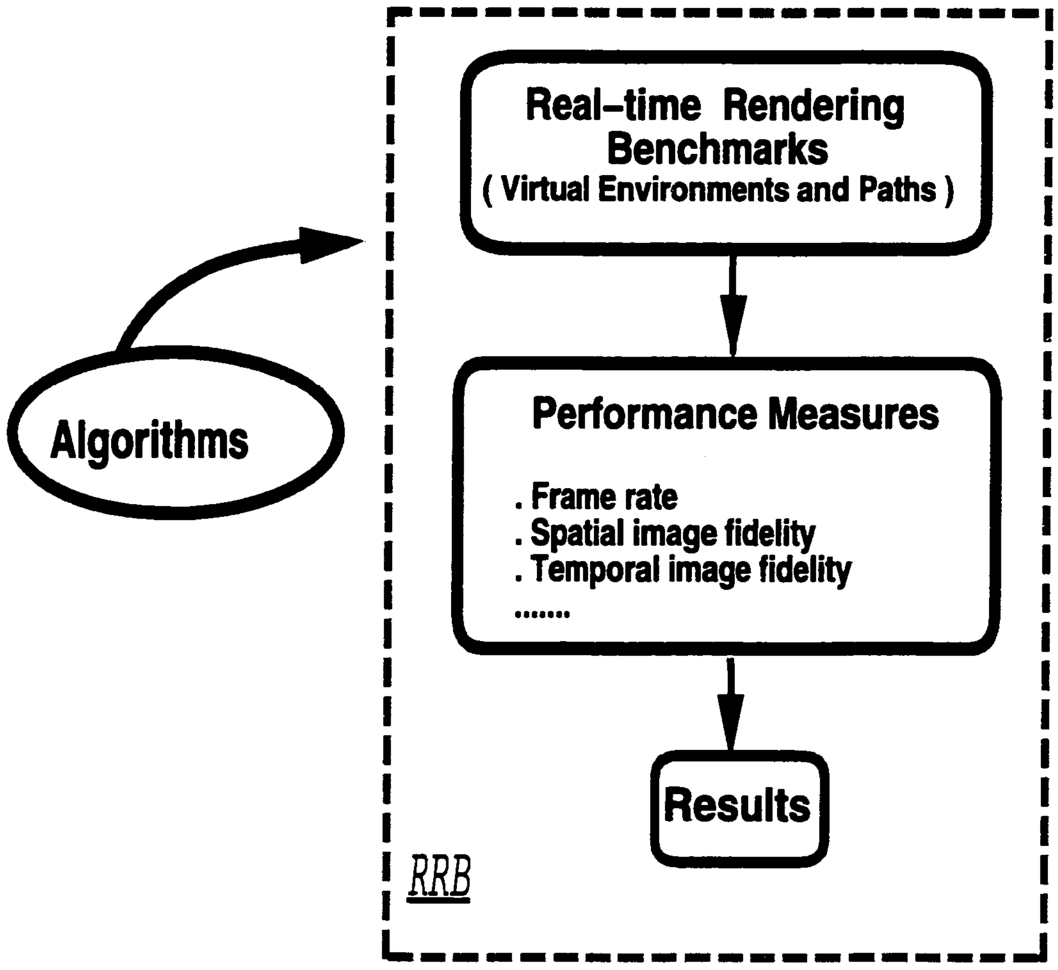

Real-time Rendering Benchmarks
(Virtual Environments and Paths)
Algorithms
Performance Measures
. Frame rate
. Spatial image fidelity
. Temporal image fidelity
....
Results
RRB

Figure 4.1: Overview of the framework

a set of metrics must be chosen. Our approach is to analyze the service offered by the algorithms and the general requirement of real-time rendering and derive the corresponding metrics. The ultimate goals of multi-resolution algorithms are to improve rendering speed and preserve the appearance of the original scene as much as possible. Improving rendering speed is crucial for a successful algorithm. All the multi-resolution algorithms are developed to at least achieve this goal. Thus, frame rate is chosen as one of the most important performance metrics. Preserving the appearance of the original models is also one of the main goals for the algorithms. It is concerned with image fidelity in both the spatial domain and the temporal domain. Spatially, the algorithms are expected to preserve the appearance of the original models from a set of viewpoints. Temporally, the transition from one frame to another should be smooth, there should be no noticeable popping artifacts and abrupt image changes. Spatial image fidelity and temporal image fidelity are two other important performance metrics in our framework. In addition, the algorithms are expected to have reasonable preprocessing time and resource consumption. In our framework, they are considered as performance metrics as well. All the metrics are used to compare the overall performance of multi-resolution algorithms.

Measures are methodologies used to implement the performance metrics. For example, frame rate is one of the performance metrics. The methods of obtaining, summarizing, and comparing the frame rate data are frame rate measures. In our framework, all the measures are developed experimentally and improved iteratively. First, the characteristics and artifacts of typical multi-resolution algorithms are observed. Then, automatic measures are developed using the advantages and also considering the constrains of real-time rendering. The measures are put into practice. If they do not yield expected results, the previous steps are iterated to improve the measures.

In the framework, the key performance measures are frame rate, spatial image fidelity and temporal image fidelity.

For frame rate, experimental and statistical methods are used to gather frame time data and compute if they satisfy a user specified threshold and how much they fluctuate during this real-time rendering process.

As we have seen in chapter 3, the spatial image distortions generated by multi-resolution algorithms are usually seen locally, around the neighborhood of simplified 3D objects. These local areas of interest are produced by choosing vertices of 3D objects. Our spatial image fidelity measure is developed to compare the local intensity differences of the local areas of interest. The vertices used to create local areas of interest are chosen automatically based on psychological studies and experimental observations in real-time rendering.

Preserving image fidelity in the temporal domain is also a key issue. For real-time rendering, the ideal situation occurs when the simplified objects look like the original ones as much as possible and the visual representations of the object changes smoothly when the user or the object moves between frames. However, multi-resolution algorithms usually simplify representations only based on the current frame. Representations of adjacent frames could be very different from each other. This leads to the discontinuities of image quality over time, that is temporal image distortions. A dynamic measure of temporal image fidelity of the algorithms is developed in the framework. It computes the transition smoothness and frequency of representation changes along a given path in a virtual environment. This measure also concentrates on the local areas of interest, as the spatial image fidelity measure does.

The key measures will be described in detail in chapter 5.

4.4 Real-time rendering testbed

As indicated in the previous chapter, real-time rendering benchmarks tend to be complex and very time consuming to build. Also, fair measurement of multi-resolution algorithms need various real-time rendering benchmarks. Therefore, instead of providing one or two standard benchmarks, the thesis concentrates on the techniques of automatically loading and running various real-time rendering benchmarks. Our approach is to provide a real-time rendering testbed. This testbed is built around two core components: virtual environments and navigation paths. A virtual environment is a collection of 3D objects, their attributes and lighting. It is loaded from VRML2 files. A navigation path is a set of viewpoints that the user uses to navigate a virtual environment. It is generated either from a 3D input device, such as a 6DOF tracker or by interpolating a set of viewpoints. By loading and combining the virtual environments and the navigation paths, various real-time rendering benchmarks are obtained for testing multi-resolution algorithms.

The virtual environment is the key component that most multi-resolution algorithms work on in real-time rendering. For example, level of detail algorithms and progressive mesh try to render simplified representations of a virtual environment to gain speed. View dependent algorithms use object hierarchy or spatial subdivisions to cull invisible surfaces in a virtual environment. In order to effectively apply these different algorithms to the same virtual environment and compare them, we represent a virtual environment with a flexible and self-complete 3D scene graph. This scene graph serves as a common software interface for multi-resolution algorithms. Various algorithms can be plugged into the testbed via this interface and perform real-time rendering tasks.

Overall, by automatically loading virtual environment and navigation path datasets, the testbed avoids the effort of building various real-time rendering tasks. It also provides a standard software interface to plug different algorithms into the standard measurement environment. The testbed makes it possible to measure and compare algorithms in a standard real-time rendering environment.

We describe the design of the real-time rendering testbed in detail in chapter 6.

4.5 Chapter summary

This chapter explains the primary goals of this research: to investigate a performance evaluation framework which is capable of automatically measuring various algorithms and producing the performance data. The fundamental components of the framework are performance measures and a real-time rendering testbed. Several typical algorithms are tested using the framework. The results and comparison of the algorithms, along with the other potential applications of our framework are described in chapters 8 and 9.

Chapter 5

Key Performance Measures

The performance evaluation framework for multi-resolution algorithms described in chapter 4 is built around two core components: performance measures and a real-time rendering testbed. The performance measures are frame rate, spatial image fidelity, temporal image fidelity, preprocessing time and resource consumption. In this chapter, we will discuss the key measures of frame rate, spatial image fidelity and temporal image fidelity in greater detail. These measures are experimentally derived from the performance requirements, common properties and typical artifacts of current algorithms which are observed experimentally. We will start from the analysis of the performance requirements and artifacts that most multi-resolution algorithms produce.

5.1 Performance requirements and artifacts of multi-resolution algorithms

Frame rate

As we have discussed, multi-resolution algorithms are developed for real-time rendering applications. The illusion of continuous motion in interactive computer graphics is usually generated by rendering a number of still images quickly. The number of frames rendered per second is called *frame rate*. In real-time rendering, the still images are generated according to the user's view point and view direction. They must render at a certain frame rate to achieve the effects of continuous and natural movement in a virtual world. Therefore, the basic requirement for algorithms is to maintain a frame rate that is above a certain threshold. This threshold can be

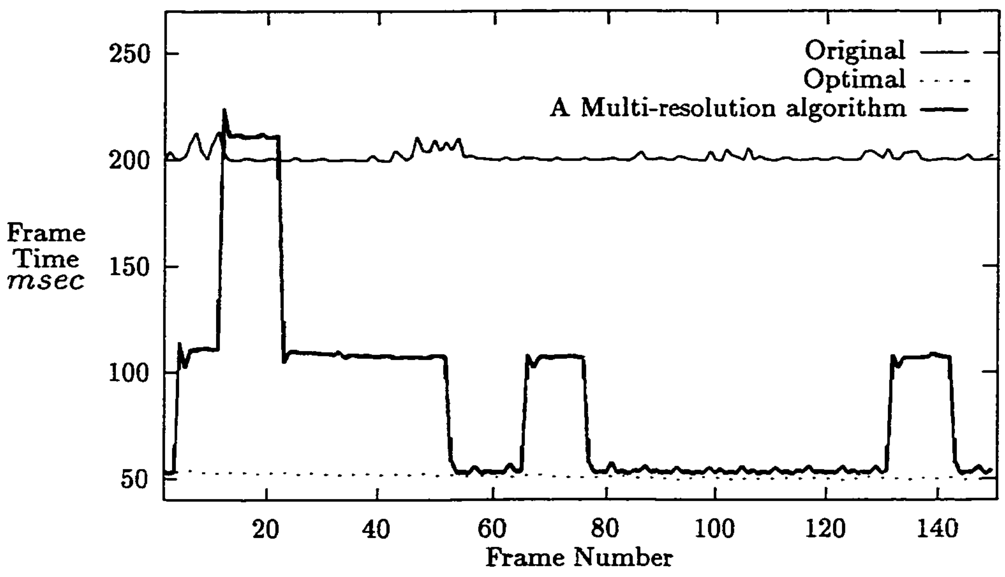


Figure 5.1: Fluctuating frame rate produced by a multi-resolution algorithm

specified by users according to different application requirements. Maintaining a constant frame rate is also very important [21]. Especially when the mean frame-time is high, fluctuations in frame-rate can influence the performance of real-time rendering tasks [60]. Therefore, frame rate and constant frame time management are important performance requirements.

Many multi-resolution algorithms don't produce guaranteed frame rate, or do not even improve frame rate. This situation happens when algorithms spend a great amount of time on their programming strategy and data management, and they use up the gain of rendering a coarser level of detail. Fluctuations in frame rate is also a common artifact of multi-resolution algorithms. Given a simple real-time rendering scenario, for example, a user walking towards or away from an object. It has been proposed that, in such a situation, the distance to the object should be considered as the criteria for switching level of detail. If the object is far away, use the coarser representation, and switch to a finer representation when the object comes closer. This type of algorithm, which does not use any time management strategies, will produce very inconsistent frame rates. Figure 5.1 shows the typical frame rate generated in one such experiment.

Spatial image fidelity

Preserving spatial image fidelity for single frames is an important performance goal for multi-resolution algorithms. To speed up frame rate, algorithms generate and render approximations of the original models. Ideally, these simplified representations should resemble the original frames generated by a non real-time rendering algorithm. During rendering, the less distortion shown in the image sequence, the better.

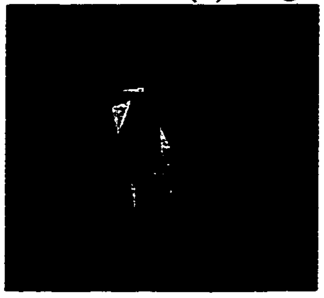
However, all algorithms produce image distortions and artifacts of some form due to simplification. Physically, these errors are differences of pixel values between the test image and original image. To users, they are of different types with distinctive features, such as loss of geometric information, loss of intensity and color information and false object positioning. These artifacts are usually seen locally, around the neighborhood of the simplified objects. They do not spread to the entire image. Figure 5.2 shows some of the typical artifacts. The bright areas in the difference images show the scale and location of these artifacts. It is evident that all of artifacts are around the neighborhood of the cow, and each algorithm produces different artifacts. Spatial image fidelity measures are developed to automatically identify the areas of artifacts and capture these artifacts.

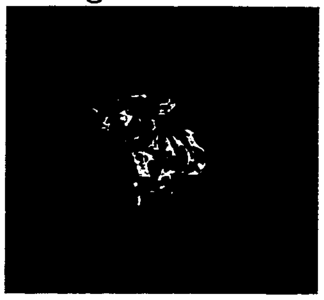
Temporal image fidelity

To preserve image fidelity in the temporal domain is also a key requirement for real-time rendering. The ideal situation occurs when the approximations look like the original object as much as possible and the visual representations of the object changes smoothly when the user or object moves between frames. However, algorithms usually simplify representations only based on the current frame. For example, in real-time rendering, an algorithm renders representation a in frame i. In the next adjacent frame $i+1$, if the frame rate, distance to the object, or other criteria passes a certain threshold, the algorithm will switch to representation b. Representation a and b in the two adjacent frames could be very different from each other. This leads to the discontinuities of image quality over time, i.e. temporal image distortions. Figure 5.3 illustrates this scenario.

These types of temporal image distortions of multi-resolution algorithms are often referred as popping artifacts. Other temporal distortions include abrupt scene changes, which are often produced by hierarchical image cache methods and view-dependent multi-resolution algorithms when they perform real-time rendering tasks. All of these temporal artifacts are also seen locally, as spatial image distortions of

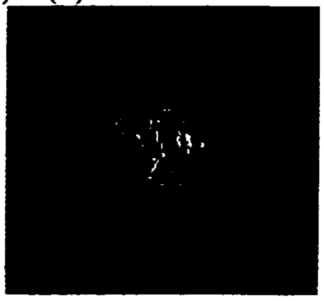
(a) original image





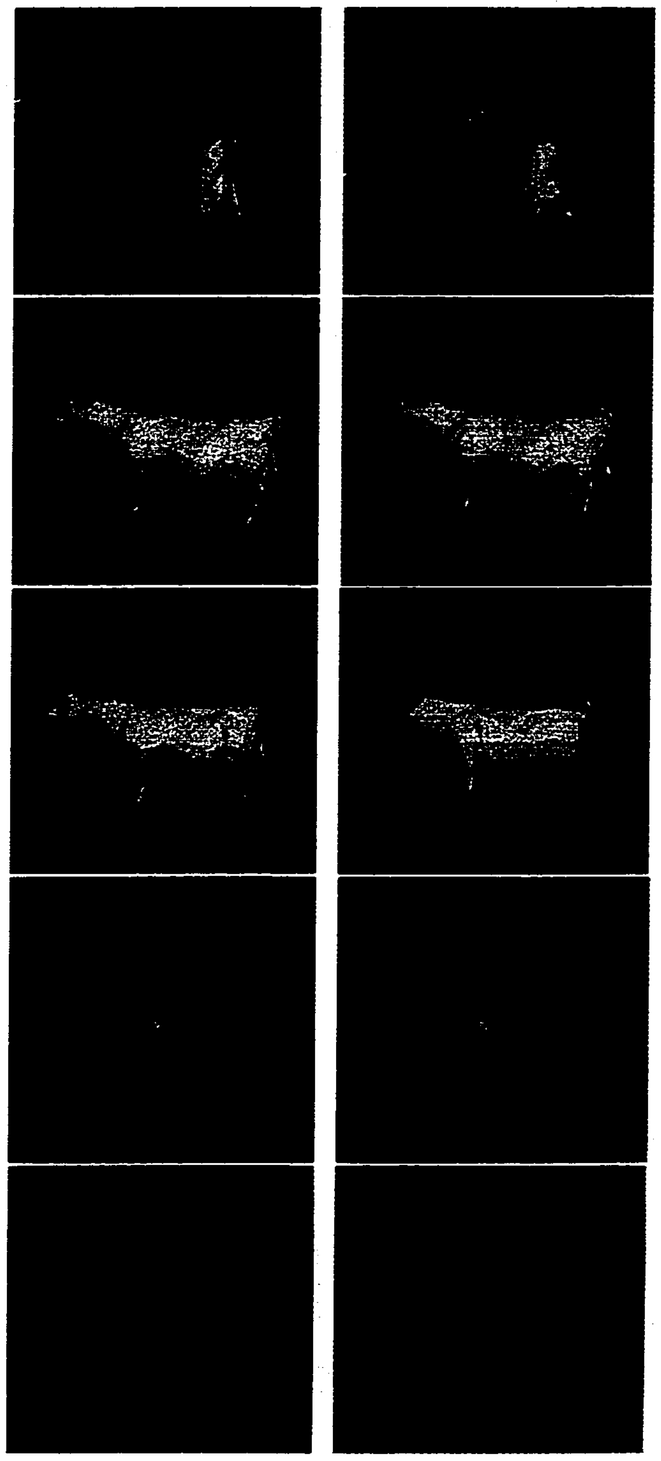
(b)simplified cow by Qslim (228 faces) (c) difference image of a and b

(d)simplified cow by Jade (228 faces) (e) difference image of a and d



(f)simplified cow by Cluster(228 faces) (g) difference image of a and f

Figure 5.2: Local image distortions generated by multi-resolution algorithms



Original images   Test images

Figure 5.3: Popping artifacts in the test image sequence, notice the difference between frame 4 and 5, frame 13 and 14

single frames.

There are two main reasons for temporal image distortions:

- Transitions between different representations are noticeable and not smooth in the context of moving viewpoints.
- Noticeable transitions between different representations happen too frequently while viewpoints are moving.

When transition roughness, and transition frequency increase, the perceptible temporal image distortion artifacts become more severe.

Previous experiments show that temporal distortions are more annoying to users than spatial distortion in real-time rendering. However, to our knowledge, no experiment has been done to measure the temporal distortion of real-time rendering algorithms. An automatic measure is required to estimate the distortions.

Other requirements

Multi-resolution algorithms are expected to use a reasonable amount of resources. However, some algorithms require large amounts of extra data which could be several times larger than the original model. It not only increases the workload of generating the data, but also involves memory management problems if the data set is too large. The model preprocessing work is also an overhead for a good real-time rendering algorithm. Extra hardware requirements, such as multiprocessors are also restrictions for general purpose real-time rendering algorithms. When the performance metrics are defined, all of these factors should be taken into consideration.

The performance requirement analysis described in the previous paragraphs implies that frame rate, image fidelity in both spatial and temporal domain, preprocessing time and resource consumption are the main performance targets for multi-resolution algorithms. Algorithms are expected to satisfy the requirement in these aspects while performing real-time rendering tasks. How well an algorithm performs in these aspects basically characterize its performance in real-time rendering. We therefore choose frame rate, image fidelity in both spatial and temporal domain, preprocessing time and resource consumption as the performance metrics in the evaluation framework. These metrics are specific and measurable. They serve as the foundation of the performance evaluators in the performance evaluation system. In our performance evaluation framework, these metrics are quantified to produce automatic objective measures. The main focus of the following section is the key measures of frame rate, spatial image fidelity and temporal image fidelity.

5.2 Frame rate

Our frame rate measures test how an algorithm satisfies the requirements of frame rate threshold and consistency. They are realized in two steps, raw data collection and data summarization.

The frame rate measures are based on the frame time of each frame f_i while an algorithm runs a benchmark. To obtain f_i, the trap instruction mechanism [32] is used. A sub-routine call is placed at the beginning of the rendering code that records the time. After finishing the rendering, another trap instruction reads the clock, subtracts the start value to obtain the elapsed time for rendering the current frame. This mechanism adds little overhead to the rendering time. The resulting data are buffered for the next step.

To compare the frame rate performance, the obtained frame time data set needs to be summarized. Various statistical methods exist for data representation and analysis [33] [37] [26]. At the moment, we are concerned with the ones that can easily be interpreted. The arithmetic mean $\overline{f}$, given by equation 5.1, minimum frame time $minf$ and maximum frame time $maxf$ are used to represent the frame rate performance. An ideal frame rate is specified as the threshold for a real-time rendering benchmark. The mean, minimum and maximum frame time in this test are compared with the threshold to see how well the algorithm maintains a certain frame rate. To compute the fluctuation of an algorithm's frame rate, mean absolute deviation, which is defined by equation 5.2, is used. Many other data summarization methods developed in statistics can also be used to obtain accurate analysis, such as sample variance and semi-interquartile range [32]. It would be useful to experiment with them in the future.

$$\overline{f} = \frac{1}{n}\sum_{i=1}^{n} f_i \tag{5.1}$$

$$F = \frac{1}{n}\sum_{i=1}^{n}|f_i - \bar{f}| \tag{5.2}$$

where f_i is the frame time of each frame.

After applying the equations to the frame time dataset, the smaller the $\overline{f}$ and F, the better the frame rate performance. For all the multi-resolution algorithms, the following baseline is applied. A real-time rendering benchmark is run with a regular rendering algorithm, that is without involving any multi-resolution algorithm. The frame rate performance produced by this regular algorithm is the baseline for frame rate performance. Any multi-resolution algorithm should achieve better performance when it performs the same task, otherwise it should be ruled out as a real-time rendering solution.

5.3 Image fidelity

All of the multi-resolution algorithms generate image distortions of some form, such as loss of geometric, intensity and color information, false object positioning and popping artifacts. These distortions are annoying to users and they won't be present if the algorithms are not forced to trade off rendering quality for speed. Our measures are developed in the context of real-time rendering [67] [68]. While an algorithm runs a real-time rendering benchmark, the measures locally compare the test image sequence rendered by this algorithm with the original image sequence rendered by regular algorithm and automatically produce the image fidelity performance in both spatial and temporal domains. The performance data can be used to generate a ranking of multi-resolution algorithms.

5.3.1 Spatial image fidelity

The requirement of preserving spatial image fidelity for multi-resolution algorithms means an algorithm should preserve the appearance of the original models or scenes from a set of viewpoints on a navigation path. However, almost all multi-resolution algorithms introduce image distortions and artifacts due to simplifications. These distortions occur locally, around the neighborhood of the simplified objects, as seen in §5.1. Our spatial image fidelity measure is used to estimate these visible distortions produced by an algorithm from a set of viewpoints.

Related measures

Image fidelity measurement problems exist in many areas, such as image processing, video signal processing, TV and monitor design, and psychophysics. A considerable amount of research has been done in these areas and many measures predicting the visual difference between the distorted image and the original image can be found in the literature.

Root mean square error(RMSE) and peak signal to noise ratio (PSNR) are widely used in image processing [24] [14] [69] [35]. Given the original image and the test image, equations 5.3 and 5.4 are used to compute RMSE and PSNR.

$$RMSE = \frac{1}{n} \sum_{(x,y)\in I0} \| I0(x,y) - I_{(x,y)} \| \tag{5.3}$$

$$PSNR = -10 \log \frac{\frac{1}{n}\sum_{(x,y)\in I0}(I0_{(x,y)} - I_{(x,y)})^2}{255^2} \tag{5.4}$$

where $\| * \|$ is the Euclidean L_2-norm. An image I is an array of n pixels (x, y), $I0_{(x,y)}$ and $I_{(x,y)}$ are the pixel's intensity in the original and test images respectively.

RMSE and PSNR are capable of measuring physical pixel-by-pixel errors between two images, however, they are also criticized for lacking local information and visual perception meanings [59]. For example, given the original image (a) and two distorted images (b) and (c) as in figure 5.4. RMSE of (b) and (c) are 8.95 and 17.58 respectively, which suggests image (c) has bigger distortions. However, as we see, image (c) looks the same as the original image, and image (b) clearly has a large local distortion. RMSE fails in this situation because it can't identify the location of the large error and it evens out the local error over the entire image.

Displaying the difference image between the distorted image and the original image can help to overcome the problem of root mean square measure. It is often used in the digital image processing community. It can indicate the location and nature of the image distortions. For example, figure 5.2 clearly shows the location and size of

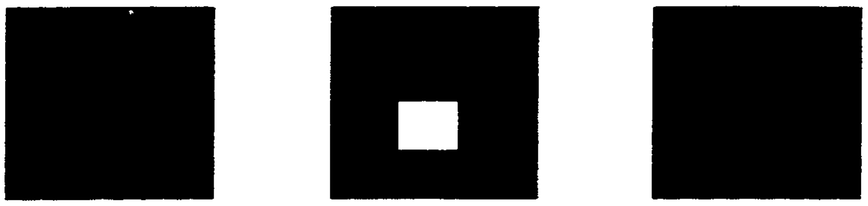

(a) original image (I = 155) (b) distorted image I=255 in the white square, otherwise I=155 (c) distorted image I= 154

Figure 5.4: Ranking the distortions using RMSE

the distortion. However, this method does not describe the distortions quantitatively, thus it can not be directly used for threshold ranking.

Another trend in spatial image fidelity research is to explore the properties of Human Visual System (HVS) and develop psychophysical models to analyze and estimate the visual difference, as in [59] [56] [16]. However, effective models have to be tested in tedious psycho-visual experiments and are often influenced by factors beyond experimental control. Also, they tend to be hard to implement and interpret.

Developing universal objective image fidelity measures is still an open problem, and is beyond the scope of this research. Techniques developed in other areas, such as image compression and HDTV must be experimentally verified for the domain of multi-resolution rendering. This is because the image distortions generated by multi-resolution algorithms have unique properties. They are new, and may have never been seen before by most users. The goal of our work is to analysis these distortions and develop specific measures to capture them. These measures serve as the first attempt to solve the problem of performance evaluation of real-time rendering algorithms. In the future, new measures can be added to the framework.

Measure Es

The spatial image fidelity measure Es automatically computes the image distortions produced by a multi-resolution algorithm while it's performing a real-time rendering task. The high-level method is as follows:

- For a given real-time rendering benchmark (see chapter6), generate the *original image sequence* by rendering the given virtual environment along the given path using a regular algorithm.

- Generate the *test image sequence* using the test multi-resolution algorithm on the same benchmark.

- The frames of the original image sequence and the test image sequence are aligned according to the given path, i.e. view points and orientations. If the test algorithm drops frames during rendering, represent the missing frames by the previous frame and make the frame number of the test image sequence equal to the original frame number. Without loss of generality, it is assumed that the first frame is never dropped.

- Compute the errors between the image pairs using equation 5.7.The average error of all frames represents the spatial image distortion generated by the test multi-resolution algorithm while it's performing the given real-time rendering task.

This method, like many related image fidelity measurement methods, is a simple image pair comparison. The critical part of a spatial image fidelity measure is how to compare each image pair. This is where Es differs from other measures.

Briefly, Es computes the image distortion in the local areas of interest only. It works in the following way:

- Identify a critical vertex, v of an object, in 3D object space.

- Obtain and record the position of v in screen space p_i for all frames while rendering the original image sequence, where i is the frame number. Mark the frame as *Invisible* if the vertex is a hidden vertex, or is out of view in the current frame. p_i is the center of one area of interest. The size of this area of interest, i.e. the number of pixels in this area is at least 1. It can be proportionally enlarged according to the number of pixels occupied by the object in the current frame.

- Compute the average of absolute intensity difference es_v between the original image and test image in the area of interest, using equation 5.5.

- Each image pair usually has multiple areas of interest. They are all obtained in the same manner as described in previous steps. The average of errors of all areas of interest accounts for the distortion of this image, which is given by equation 5.6.

- Equation 5.7 defines the overall distortions generated by the test algorithm while it performs the real-time rendering benchmark.

$$es_v = \begin{cases} 0 & \text{if v is Invisible} \\ |I0 - I| & \text{otherwise} \end{cases} \tag{5.5}$$

where $I0$ is the average intensity of the area in the original image and I is the average intensity of the area in the test image.

$$Es_i = \frac{1}{n_v} \sum_{v \in V} es_v \tag{5.6}$$

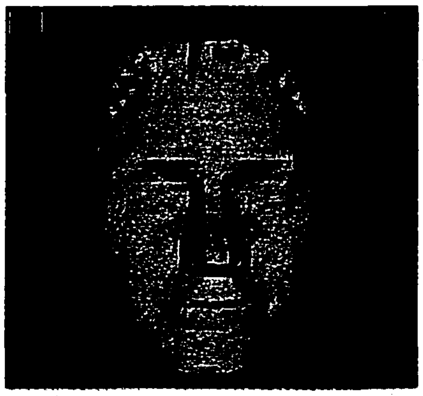
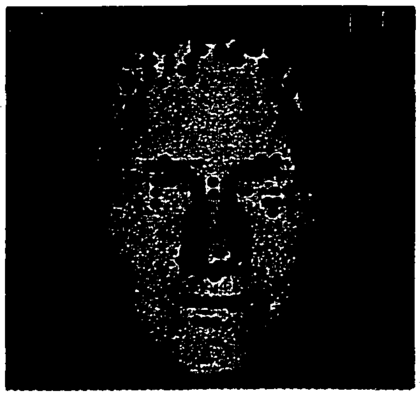
where n_v is the number of areas of interest, and V is the set of critical vertices.

$$Es = \frac{1}{n} \sum_{i=1}^{n} Es_i \tag{5.7}$$

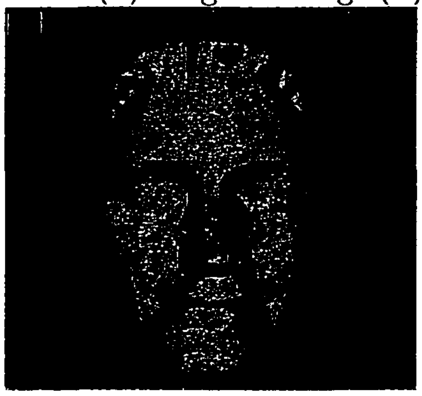
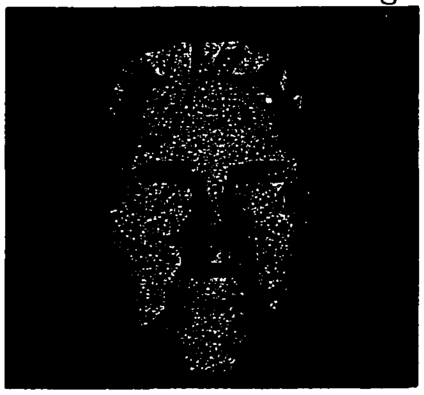
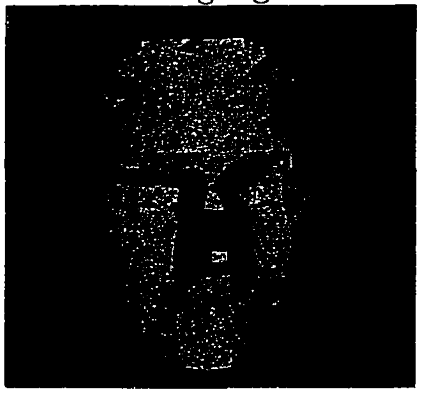
where n is the number of frames of the original image sequence.

The value of Es is proportional to the absolute spatial image distortions produced by an algorithm. That is, a large value indicates that the visual representations produced by the algorithm are significantly different from the optimal representations for the current real-time rendering benchmark.

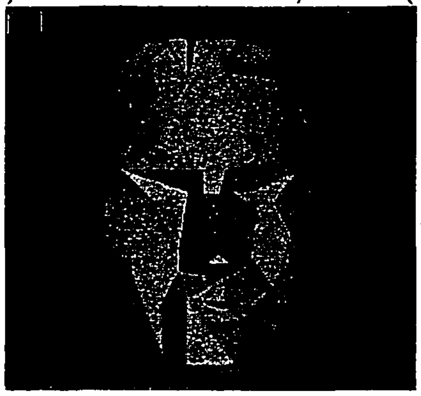
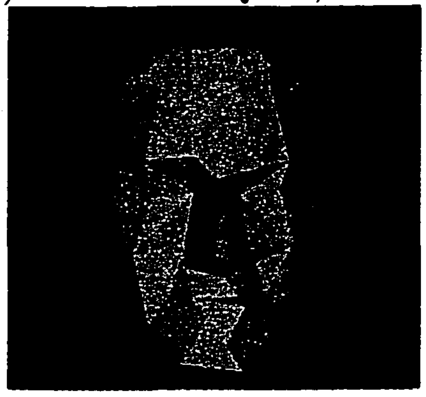
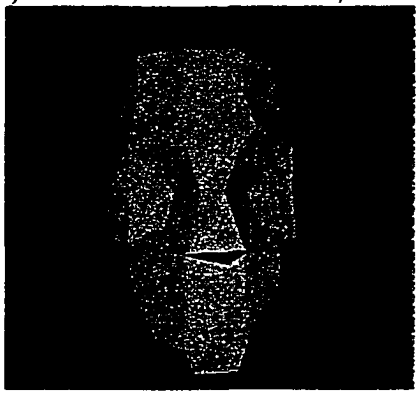
Overall, Es computes the images distortions locally, in the areas of interest. It takes advantages of real-time rendering and multi-resolution algorithms. First, in real-time rendering, the local area of interest of images can be obtained directly from 3D object space and kept track of by Es. This information is crucial for successful image fidelity measures. Most other image fidelity measurement problems are difficult because these error locations can not be easily identified from image space. Second, multi-resolution algorithms produce image distortions of unique properties. They are only seen around the neighborhood of simplified objects. and do not spread to the entire images. Es can just focus on these local areas.

(a) Original image (b) Measured areas shown in green and blue highlights

(c)Es = 1.96648 Jade, 1050f (d)Es = 5.04542 Qslim, 1043f (e)Es = 18.4634 Cluster, 1043f

(f)Es = 18.605 Jade, 190f (g)Es = 22.1036 Qslim, 193f (h)Es = 23.7162 Cluster, 190f

Figure 5.5: Es on single frames

Figure 5.5 demonstrates some experimental results for Es. More results and analysis are reported in chapter 8.

In figure 5.5. the green and blue highlights are the areas of interest. It is known from previous discussions that these areas are around the neighborhood of chosen vertices in 3D object space. It is natural to ask how to choose these critical vertices In this framework, the critical vertices are chosen automatically. They are mainly extrema of curvature or mesh discontinuities. The choice of critical vertices is discussed in greater detail in §5.3.3. For the moment, and also for the next section on temporal image fidelity, we assume the critical vertices have already been selected.

5.3.2 Temporal image fidelity

Ideally, multi-resolution algorithms should preserve temporal image fidelity. In an image sequence rendered by an algorithm, transitions between different simplified representations are expected to be smooth, and noticeable transitions should occur as little as possible. However, most multi-resolution algorithms simplify representations only based on the current frame. They often produce temporal errors, such as popping artifacts. The temporal image fidelity measure Et dynamically measures these temporal distortions while an algorithm runs a real-time rendering benchmark.

Measure Et

Et uses the same high-level mechanism as Es, in that it computes the temporal image distortions by comparing the original image sequence and test image sequence. In Et, both original and test image sequences are described as discrete functions of frame number. They are aligned with respect to view points and orientations, just as in Es. Ideally, for a temporal image measure, the original and test image sequences should be represented as functions of time. Temporal image fidelity should be measured in the context of absolute time as well. However, on one hand, the original image sequences can not be generated in real-time, otherwise there would be no need for real-time rendering. On the other hand, multi-resolution algorithms usually do not produce a constant frame rate, as discussed in §5.1. To align images with respect to time, interpolation or blending of images is needed. The interpolation of images across time without impairing perceivable image quality is still an open research problem. Therefore, temporal image fidelity is only computed in the context of image sequences. The artifacts of inconsistent display rate are evaluated by the frame rate measure described in §5.2.

Et is for estimating the temporal error, i.e. popping artifacts in a given test image sequence. As we discuss in §5.1, the temporal distortions are the accumulated representation differences between adjacent frames. *Et* computes the differences in the following way:

- For a given original image sequence and test image sequence, obtain and keep track of a local area of interest in the same manner as in *Es*.(§5.3.1)
- For each pair of images from the test image sequence and original image sequence, compute the intensity error e_i of the local area of interest using equation 5.9.
- Compute the average of the deviation of the intensity error differences between two successive frames in the image sequence, as given in equation 5.8. The result is the temporal error of this local area of interest in the test image sequence.
- The temporal error for the test image sequence is the average of Et_v at all local areas of interest, as defined in equation 5.10.

$$Et_v = \frac{1}{n}\sum_{i=1}^{n-1} |e_{i+1} - e_i| \quad \text{if } I0_i \neq 0 \text{ and } I0_{i+1} \neq 0 \tag{5.8}$$

$$e_i = \begin{cases} I0_i - I_i & \text{if } I0_i \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

$$Et = \frac{1}{n_v}\sum_{v \in V} Et_v \tag{5.10}$$

Differences between the visual representations at the local area of interest in adjacent frames contribute to the values of Et_v. The larger the difference, the larger the value of this measure will be. If an object changes representations frequently, and these representations are significantly different, then Et_v will have a large value. This situation corresponds to an object that pops noticeably and frequently as the user moves through the environment. The average of Et_v at all local areas of interest represents the temporal distortions of an algorithm.

Analysis of measure Et

Et is based on the deviation of vertices' intensity error between adjacent frames. In real-time rendering, a vertex color is determined by its position, normal, material and lighting in the scene. Theoretically, or according to the OpenGL lighting model [63], it is independent of viewpoint if there are no moving lights, spotlights and specular reflection effects in the scene. In this case, the vertex color of the original model remains the same as long as it is visible. The intensity of the corresponding position in the test image sequence should maintain the same value if there are no temporal artifacts in the test image sequence. If there are any intensity changes at the corresponding position in the test image sequence, they are purely caused by the algorithm's temporal errors, such as popping artifacts. Et accurately represents this fact in the ideal situation.

In the cases with spotlights, moving lights or specular reflection in the real-time rendering task, the intensity of a vertex changes with moving viewpoints. Thus, the intensity at the same object position doesn't stay the same from frame to frame, even for the original image sequence. So it doesn't in the test image sequence. Therefore, there will be a certain amount of noise if we directly compare the intensity deviation of adjacent frames.

Thus, Et uses intensity error e_i between the test image and the original one for each frame pair. This tries to reduce the noise and isolates the visual distortions caused by the multi-resolution algorithm in the measured area from normal intensity changes from frame to frame. It does not claim to be the best solution. However, in practice the results turn out promising, as we will see in chapter 8.

Since Et is based on the pixel intensity of vertices, its accuracy can intrinsically be affected by another type of noise – aliasing. For example, in the ideal case of no moving lights, spotlights and specular reflection effects in the scene, the intensity of a vertex should maintain the same value in an image sequence so long it is visible.

However, due to limited sampling rate and computation precision, the intensity can vary among different frames. This will cause *Et* to have a non-zero when there are no popping artifacts at all. *Et* is not capable of removing such noise completely. However, in practice, algorithms are measured using the same real-time rendering benchmarks, and the noise tends to follow a similar pattern. It thus does not impose a serious effect on ranking algorithms temporal image fidelity. The experiment results in both chapter 8 and chapter 9 provide the evidence.

5.3.3 Critical vertices

As we have seen in the last two sections, both of the spatial and temporal measures are based on the intensity difference in the local areas of interest. These local areas are obtained from the chosen vertices in the given 3D scene. In this framework, these vertices are termed *critical vertices*. They are the source of the local areas of interest. This section examines how to select these vertices automatically.

A 3D scene consists of a set of 3D geometric objects and their attributes. Any geometric object can have many vertices. For example, the "cow" used previously has 2904 vertices. The "Stanford bunny" shown in figure 5.6 has 35947 vertices. In real-time rendering, many of these vertices may be invisible in many frames due to visibility culling. Even if they are visible, thousands of vertices can be projected to just a few pixels. If all of them are used to compute local area of interest for every frame, it is computationally inefficient and unnecessary. It is evident that critical vertices need to be selected from the vast number of vertices in a 3D scene.

The problem is which vertices of a 3D object are critical vertices? A classic result on this issue was reported by Attneave in 1954 [3].In his famous visual perception experiments – the Attneave Cat, Attneave discovered that the most informative points of a 3D object are at its extrema of curvature. Changes in these areas are more noticeable to users than other areas. Previous research on mesh simplification have also found similar results for this problem [28] [23]. It has been reported that surface discontinuities of a 3D object such as open boundaries, and borders between differently colored regions, and creases are often among its most visually significant features. The appearance of their neighborhood should be preserved first.

The above discoveries suggest that vertices on extrema of curvature and discontinuities of a 3D object are a good choice for critical vertices. We therefore start the experiments with this idea. The vertices of an object are classified into three categories, extrema of curvature, discontinuities and regular vertices. Critical vertices are mainly chosen from the first two categories.

Extrema of curvature

On a surface, there is a curvature in every direction. Unless the curvature is equal in all directions, there must be a direction in which the normal curvature reaches a maximum and a direction in which it reaches a minimum. These directions are called the principal directions and the corresponding curvatures k_1, k_2 are the principal curvatures. In practice, the principle curvatures are often used to define other curvatures to represent surface properties, such as the Gaussian curvature $K = k_1 * k_2$ and the mean curvature $H = (k_1 + k_2)/2$. When computing extrema of curvature, only the principle curvatures are considered.

The extrema of curvatures on a surface occur where there is the curvature maxima or minima. Curvature can be positive and negative. So, curvature extrema refer to where the magnitudes of the principal curvatures are maxima or minima. Users can specify the range of the principle curvature magnitude required for a point to qualify as an extremum.

Various methods are available to compute k_1 and k_2 of a vertex on a discrete surface, i.e. a mesh. We found Garland's quadric matrix is preferable [22]. He defines a quadric matrix Q for each vertex in a given mesh. Q is the sum of a set of fundamental quadric matrixes which represent an entire set of planes adjacent to the vertex. The vertex is the corner of this set of planes. Equations 5.11 and 5.12 define the quadric matrix Q.

$$Q = \sum_{p \in planes(v)} K_p \tag{5.11}$$

$$K_p = \begin{pmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & db & dc & d^2 \end{pmatrix} \tag{5.12}$$

where ax + by + cz + d = 0 is the plane equation of an adjacent face of the vertex v. Garland [23] has proved that for a sufficiently dense mesh, the eigenvalues of the quadric matrix Q of a vertex are proportional to the squares of the principal curvatures and that its eigenvectors are the corresponding principal directions. So, for each vertex on a mesh, The principle curvatures k_1 and k_2 are estimated with Garland's quadric matrix Q. In detail, the eigenvectors and eigenvalues are computed using Jacobi's algorithm. Sort the three eigenvalues, the second one a_2 is proportional to k_1^2 and third value a_3 is proportional to k_2^2. The vertex is an extrema of curvature if the larger one of a_2 and a_3 is larger than a user specified threshold.

Using this method, we might miss some extrema points. When the principle curvatures of all vertices in an object are larger or equal to zero, the vertices of minima of curvature are missed. If the principle curvatures of all vertices in an object are smaller or equal to zero, the vertices of maxima of curvature are missed. However, in practice, these situations seldom occur for a complex 3D model.

Surface discontinuities

The geometry and attributes of a 3D object can give rise to discontinuities in its visual appearance. The discontinuities are of three types. They are geometric open boundaries, borders between differently colored regions and creases. They usually form visually significant features of 3D objects.

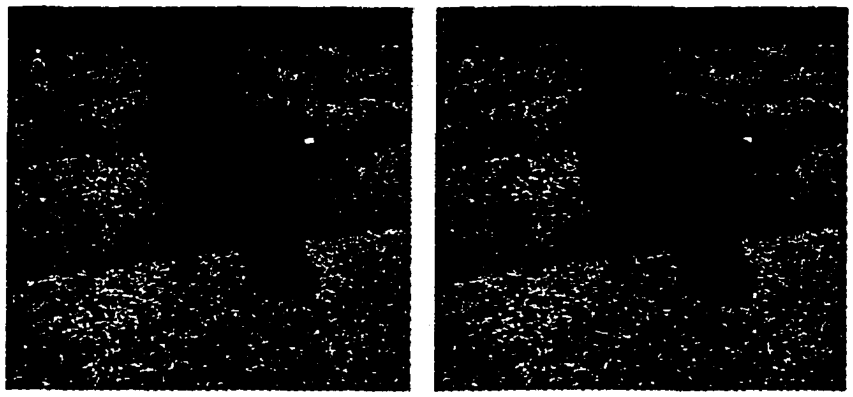
Computing the discontinuity of a mesh is trivial and straightforward. Typical methods can be found in many computer graphics books, such as [19].

Figure 5.6 show the critical vertices of sample objects selected using the above described method.

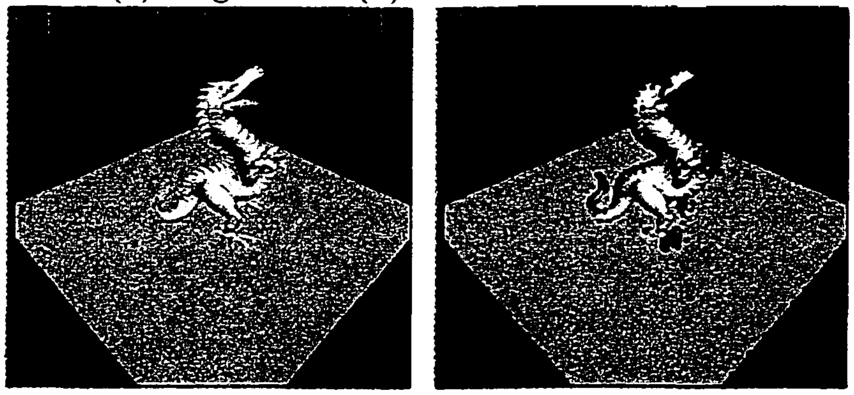
5.3.4 Size of the area of interest

The image fidelity measures of Es and Et are based on image pair comparison at the local areas of interest. Previous sections explained how to find the location of an area of interest. This section briefly discuss a method of computing its size, i.e. the number of pixels in the local area of interest.

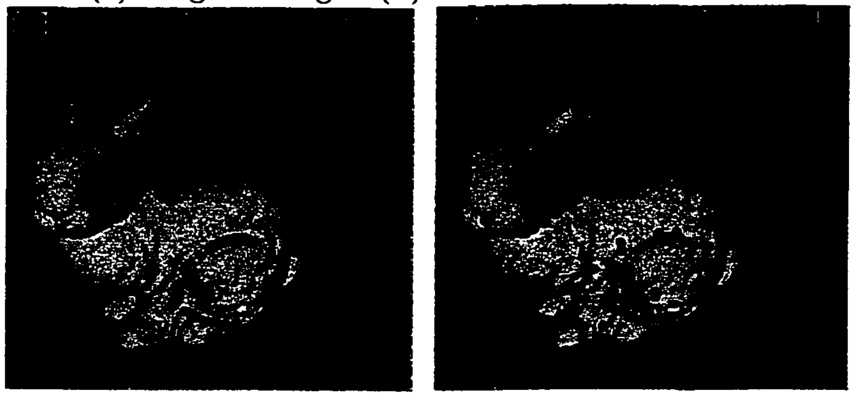
By default the size of an area of interest is one pixel, the pixel corresponding to the critical vertex in the projection of the object. Previous research in real-time



(a) Original cow (b)Visible critical vertices



(c) Original dragon (d)Visible critical vertices



(e) Original Stanford bunny (f)Visible critical vertices
Green highlights are extrema of curvature, blue highlights are mesh discontinuities

Figure 5.6: Critical vertices

rendering, such as [21], indicates that a larger object in screen space is more visually important than a smaller object. This suggests that the size of a local area of interest should be selected to reflect the visual importance of this area. The larger the object appears in the screen in real-time rendering, the larger the local areas are. Equation 5.13 shows that the size of a local area of interest N_i is proportional to S_i, which represents the screen size of the associated object, where k_1 is a user determined constant.

$$N_i = k_1 * S_i \tag{5.13}$$

The problem is how to compute the scalar factor S_i. For a simple virtual environment which has only one object and a black background, computing the exact number of pixels occupied by a 3D object is easy. For each frame, it is the entire window size minus the number of background pixels. However, for a more complex virtual environment which has multiple objects or a textured background, it becomes a non-trivial problem. Image analysis techniques are required to compute the pixel number of an 3D object in image space. In the context of our image fidelity measures, we are not concerned with the exact screen size of an object, rather an approximation. S_i is not necessarily the exact screen size of an object. It is sufficient to be a proportional scalar factor which represents the changing screen size of an object. More importantly, we prefer an efficient method to compute S_i since this computation is needed for every frame of a real-time rendering task. To satisfy these two requirements, we choose an simple method to obtain the approximation from the 3D object space. A 3D object's bounding box and the distance between the user and the object contribute to its screen size. The larger the object is, the larger area it occupies in an image; The closer an object is to the viewer, the larger it appears. Based on this projection principle, S_i can be defined in the equation 5.14:

$$S_i = k_2 * B/d_i \tag{5.14}$$

Where k_2 is a user defined constant, B is the diagonal of object bounding box and d_i is the distance between the viewer and the object in frame i.

This method basically reflects the idea that a larger and closer object has large areas of interest. It is computational efficient, but it is by no means the best solution for computing the size of an local area of interest. It can't model the fact that an irregular shape has different screen size projected from different directions. Accurate and efficient methods should be studied in the future. Another factor in computing the size of an local area of interest is the constants k_1 and k_2. They determine the range of the actual pixel numbers in a local area of interest. Our experience is that they should be small enough to keep the actual pixel numbers in the range of 1 to 10. Large sizes can cause many overlaps of the local areas. They can also introduce intensity errors at the object boundaries. Therefore, Es and Et work better in practice if the local areas of interest are small.

5.4 Chapter summary

The key performance metrics and automatic measures of multi-resolution algorithms for real-time rendering have been presented. The metrics are defined based on the general requirements that the algorithms are expected to satisfy. The typical artifacts of multi-resolution algorithms are also analyzed in this chapter. Finally, the automatic measures of of frame rate, spatial image fidelity (Es) and temporal image fidelity (Et) that are developed to capture the artifacts and evaluate the algorithms are presented. The measures are designed to work specifically for multi-resolution algorithms in real-time rendering applications. They are easy to implement and interpret.

Chapter 6

Real-time Rendering Testbed

As we have seen in chapter 3, many multi-resolution algorithms have been proposed, such as level of detail, continuous multi-resolution and view-dependent multi-resolution algorithms. The goals of these algorithms are to improve rendering speed and preserve image fidelity as much as possible when applied to real-time rendering applications. To effectively evaluate these algorithms, a unified test environment is needed. First the general requirements of real-time rendering benchmarks are discussed. Then, the real-time rendering testbed design is described, along with how the testbed can load various real-time rendering benchmarks and also plug in algorithms for measurement. Lastly, the parameters and factors of the real-time rendering benchmarks are discussed.

6.1 Real-time rendering benchmark requirements

Loosely speaking, any interactive 3D application can be used as real-time rendering benchmarks. They include 3D navigation, interactive 3D operations, and real-time CAD. Among them, 3D navigation is the most basic and common real-time rendering task. It enables a viewer to experience a virtual environment by simulating a walk through of the computer generated 3D models. They can stand alone, or they can form the basis for other real-time rendering applications, such as real-time CAD/CAM. In the performance evaluation framework we concentrate on 3D navigation applications only. A broad range of 3D navigation applications of different workloads are used as real-time rendering benchmarks for testing multi-resolution algorithms.

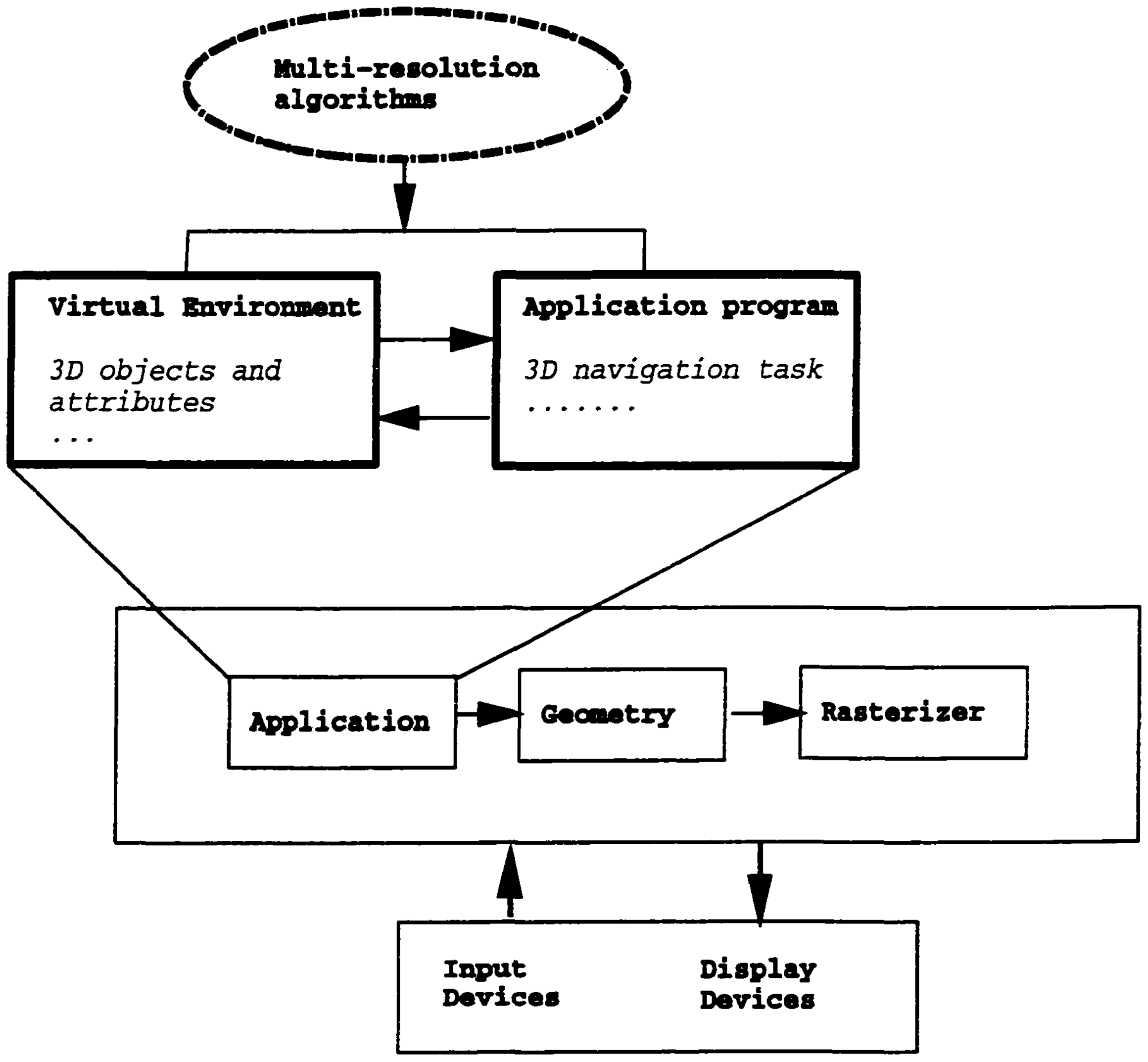


Figure 6.1: 3D navigation application system

Building a 3D navigation system, as building other interactive 3D applications, tends to involve many hardware and support software configurations. Besides the rendering pipeline (§3.1), input devices and display devices need to be configured as well. Figure 6.1 illustrates the high-level conceptual diagram of such a system.

From an application point of view, the following has to be done to obtain one real-time rendering benchmark:

- Model the geometric objects in the virtual environment
- Define navigation tasks.
- Create a view at each step of the navigation.

- Render the virtual environment using the graphics rendering pipeline.
- Configure input devices and display devices

Development of each of the above steps is time consuming and not easy. Particularly, large and complex virtual environments are notoriously difficult to model and very tedious to build. To evaluate multi-resolution algorithms thoroughly and fairly, one would like to have many different real-time rendering benchmarks. It is obviously not optimal for users to design every benchmark step by step from scratch. This raises the first requirement for real-time rendering benchmarks. One would like to have high-level system support to avoid complex system configuration, and systematic methods to simplify real-time rendering benchmark design. It would be ideal to automatically load and run various real-time rendering benchmarks.

The second requirement of the benchmarks is to have a standard software interface for the algorithms evaluated in the real-time rendering benchmark. On one hand, it is easy for a multi-resolution algorithms to be hooked up and tested with many benchmarks without changes. Thus the algorithm can be evaluated more thoroughly and fairly. On the other hand, the standard interface makes it possible to test diverse algorithms with the same benchmarks and thus compare their performance. When we look at the system architecture of a typical real-time rendering benchmark (Figure 6.1), we see that multi-resolution algorithms only interface with a small portion of it, despite its complex system configuration. Algorithms are mainly concerned with the application module. In particular, most multi-resolution algorithms work on virtual environments only. They try to choose an appropriate representation of the virtual environment and produce faithful pictures of it interactively. For example, level of detail and continuous multi-resolution modeling try to render simplified representations to gain speed. View-dependent multi-resolution algorithms also use an object hierarchy or spatial subdivisions to cull invisible surfaces. Thus, with a well designed system architecture for real-time rendering benchmarks, a software interface to virtual environments is essential and enough for joining algorithms and benchmarks together.

Thirdly, real-time rendering benchmarks have many parameters, including system configuration parameters and task parameters. These parameters should be recorded and easy to control. Thus they can be used repeatedly to benchmark a set of different algorithms and across diverse real-time rendering application environments.

Overall, these requirements indeed ask for a systematic unified real-time rendering testbed. It avoids complex system configuration, simplifies the work of benchmark design, loads and runs various real-time rendering benchmarks, and also allows algorithms to be plugged in, measured and compared. It is also one of the primary research goals to develop a unified benchmarking platform to satisfy the above requirements. The next section describes the design approach of the testbed.

6.2 Testbed design

The central task of the testbed design is to capture the general characteristics and control structures of typical real-time rendering benchmarks and build a real-time rendering application framework. As we have discussed, real-time rendering benchmarks simulate users' viewing while they are walking through, observing or examining a 3D virtual environment. Typical examples include examining a set of 3D models from different viewpoints in real-time, walking through a virtual building, flying in a virtual outdoor scene, or wandering or even jumping in any computer generated world.

Such applications can almost all be implemented in the following simulation loop.

- Obtain a viewpoint on a navigation path, either from user input or program simulation.
- Update the virtual environment based on the viewpoints.
- Render the virtual environment.

From a system analysis point of view, such an application can also be divided into two main components :

- Virtual Environments: They are composed of hierarchically grouped 3D geometric objects, their properties and behaviors. The properties include materials, textures, transformation hierarchy and lighting. A basic object behavior is to have its graphics output on the screen through the rendering pipeline.

- Interactive Navigation Paths: They represent key parameters of 3D navigation programs. A navigation path consists of a sequence of viewpoints and view orientations.

The goal of the framework design is to encapsulate the routine simulation loop in the two components and provides a high-level software interface for the algorithms. In the mean time, by loading various virtual environments and paths, and mixing and matching them, the framework can run a variety of real-time rendering benchmarks which facilitate performance evaluation of multi-resolution algorithms.

6.2.1 Virtual environment

The virtual environment is the most important component in the testbed, since it is where most algorithms interact with our framework. We are concerned with providing a standard interface for the algorithms and also various virtual environments with rich content. It raises the technical issues of scene graph, virtual environment dataset and file parser.

Scene graph

A virtual environment is a collection of geometric objects, their properties and lighting in a 3D world. It has many parameters, such as object coordinates, edges, faces, object size, material, texture, lighting, group and transformation information etc. The most popular modeling structure for describing such virtual environments is hierarchical modeling. In this technique, a virtual environment is organized as a directed acyclic graph (DAG), which is also referred to as a *scene graph*. The nodes in the graph hold graphical data and dictate grouping structure, while the edges of the graph describe the inheritance relationships of state information. Figure 6.2 and 6.3 show an example of the scene graph and view of a virtual environment.

A scene graph not only organizes its constituent objects, but also defines their regular rendering protocol. Its architecture is of great interest to multi-resolution algorithms, especially view-dependent algorithms. A number of scene graph architectures exist. A typical one in the early years is the *central structure storage* in PHIGS [19]. It stores the objects in a structure hierarchy which facilitates sequential display

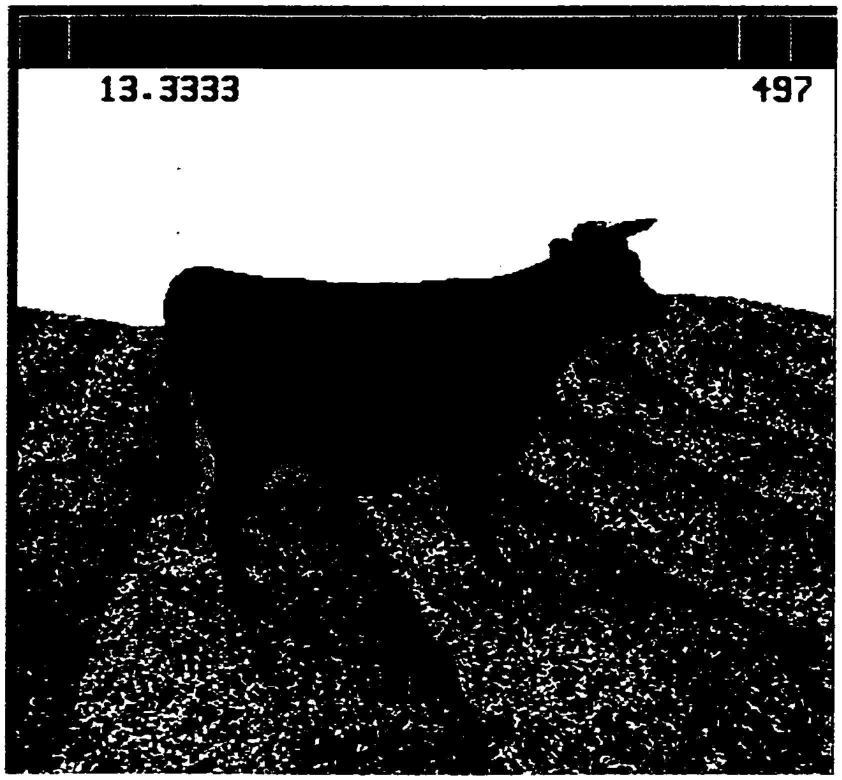


Figure 6.2: Cow picture

traversal. However, this has a major limitation. In the scene graph, a leaf node's state (attributes or transformation) is affected by any state changes of a node above or to the left. In the example shown in figure 6.3, if it was structured in PHIGS or OpenInventor scene graph, the cow has to be displayed before the terrain to get a correct picture. Therefore, the rendering traversal of such a scene graph has to be from top to bottom and left to right. This characteristic limits application of many current multi-resolution algorithms, such as view-dependent and hierarchical cache algorithms. Many current scene graph structures share the same limitation, such as SGI's OpenInventor and the early version of VRML 1.0.

To provide a standard and usable interface for existing multi-resolution algorithms, one would like to have a scene graph structure without the previous limitation. An algorithm should be able to traverse the scene graph in whatever order it wishes. It can traverse the scene graph from left to right and top to bottom, in level order from right to left, or even in parallel. This requires a scene graph structure, in which a leaf node's state is only defined by the nodes in a direct path between the scene graph's

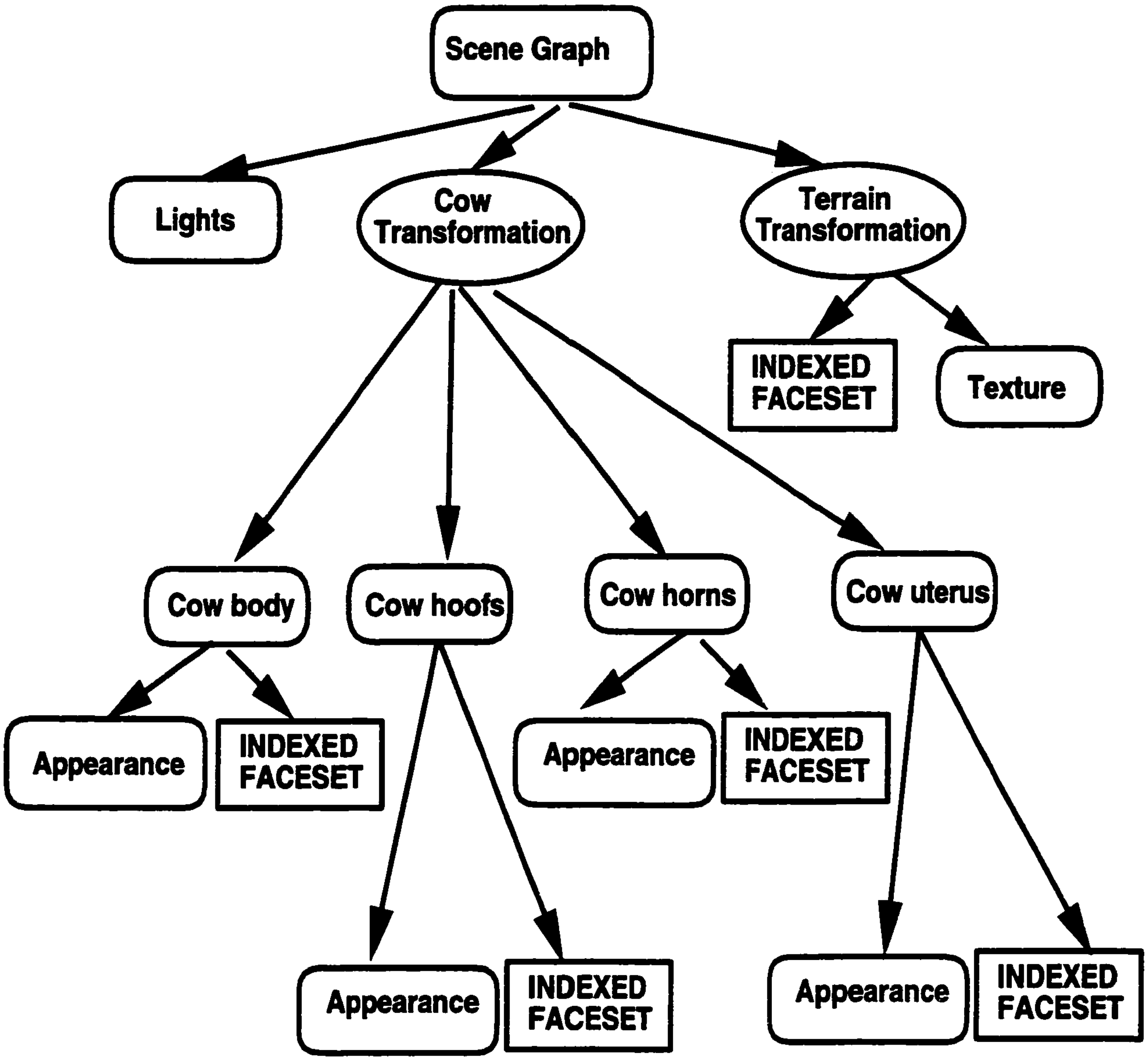


Figure 6.3: Cow scene graph

root and the leaf. The scene graph structure in our performance evaluation framework satisfies this requirement. It is defined as a DAG where the nodes in the graph hold graphical data and dictate grouping structure, while the edges of the graph describe information inheritance relationships. A scene graph may contain internal nodes and leaf nodes. The leaf nodes hold geometric and appearance information. In our system, they are *triangular indexed face sets* , i.e. meshes, and their appearance properties. This is the node which most current multi-resolution algorithms interact with. Since algorithms mainly work with meshes, they are the main geometry nodes defined in our scene graph. Internal nodes of the scene graph hold either grouping information or scene state information, such as transformations. The grouping and scene state information can only be inherited along the path from root to leaf. Thus, there is no restrictions on how an algorithm traverses the scene graph. Virtual environments are built upon this relatively simple data structure, so that a number of algorithms can be applied to it and evaluated. Figure 6.4 shows the scene graph architecture.

This scene graph is similar to the scene graph structure of Java 3D and VRML 2.0, in the sense that it does not restrict the display traversal order. However, it is much simpler than Java 3D and VRML 2.0. It serves one purpose only, which is to support plugging in various multi-resolution algorithm. It is intended to be minimal and complete to reduce the work load of the performance evaluation system itself. In contrast, the Java 3D API is designed to provide a rich set of features for creating interesting 3D worlds in Java using a high-level object-oriented programming paradigm. VRML 2.0 is designed for interactive 3D graphics on the web. The scene graph's Internet functionalities are certainly not of interest to our problem. Both Java 3D and VRML 2 scene graphs have complex and huge structures. We have experimented with the VRML 2.0 scene graph. The file parser code itself and the generated scene graph from a medium size 3D scene consume a huge amount of memory. On a networked SGI platform with 32M RAM, page swapping occurred during the actual performance evaluation. Therefore, a simple, minimal and self-complete scene graph structure was designed and implemented as shown in figure 6.4 for our performance evaluation framework.

In the scene graph, each node encapsulates its basic behavior, which is display its graphics output on the screen. Together they define the regular rendering algorithm.

Virtual environment dataset

It is tedious and very time consuming to model interesting virtual environment. This conclusion is drawn from other researchers' and our own experience. In 1996, we did an "Athabasca Hall" walk-through project. It took us one whole month to model the 3D Athabasca Hall interior and exterior from its 2D blue print. Fair measurement of multi-resolution algorithms requires many virtual environment models. It is not productive for us to spend a huge amount of time on modeling them all.

Many 3D models are available on the Internet, from single objects to complex virtual environments. They are often designed for interactive 3D graphics applications in real-life. These models are good resources for composing the virtual environments. They can be used directly or they can be put together to construct virtual environments of different scales and complexity. In either way, it alleviates the user's tedious effort of virtual environment modeling. On the Internet, the models are often found in different file formats, such as 3DS, DXF, OpenInventor, VRML 1.0, or VRML2.0. Most of the formats are interchangeable. In our framework, we use VRML 2.0 as external data format. All the virtual environments are described in VRML 2.0 format and follow the VRML2.0 syntax.

VRML 2.0 parser and scene graph generator

VRML 2.0 is a scene description language that describes the geometry and behavior of a 3D scene or world. It is the most commonly used file format for web 3D graphics currently. It has the capacity of describing complex and animated 3D worlds with its scripting feature. The VRML 2.0 specification and examples can be found on the web [10] [11] and the VRML reference manual[4]. Several VRML 2.0 browsers have been developed commercially, such as Cosmo Player [64]. Unfortunately, the implementation details and source code are not available to us. In order to use the VRML 2.0 datasets, a prototype VRML 2.0 parser and scene graph generator was implemented. They are able to parse any VRML 2.0 file, filter out un-wanted information and generate an in-memory scene graph of the structure described in section 6.2.1. The VRML 2.0 parser and scene graph generator bridge various existing VRML2.0 worlds with our scene graph structure, thus solving the problem of virtual environment modeling.

6.2.2 Interactive navigation paths

Some multi-resolution algorithms [21] [50] lack complete evaluation in that they were tested in just one or two fixed-path navigation experiments. These paths are all artificially generated. To test the algorithms thoroughly, we need many more different interactive navigation paths, both mathematically generated paths and the ones captured from users' natural motions. We denote the mathematically generated path as *level 1 paths* and the natural movement generated path as *level 2 paths*. The level 1 paths are used to evaluate the algorithm for some particular task, such as moving along a path with high path coherence, or with little path coherence. They are also good for some special tests, such as how the algorithm works while viewers are moving towards or away from the geometric objects. The level 2 paths are for testing how algorithms behave while running real life navigation tasks.

The challenge is to provide techniques to generate, record and apply these interactive navigation paths to virtual environments. The level 1 paths are constructed by defining and interpolating the key points of a path using a mathematical representation, such as walking along a B-spline path. The interpolation methods include linear interpolation and cubic spline interpolation. Linear interpolation is usually fast to compute. The continuity is not as good as cubic spline interpolation. However, cubic spline interpolation is computationally expensive. The level 2 paths simulate users' natural motions. For example, a user holding a 6DOF tracker can freely navigate a virtual environment in whatever way she wants, such as any combination of looking up, down and around, walking or flying in 3D space. Such a path can be collected from any 6DOF interaction device, such as head mounted display, 6DOF trackers, space-balls and joysticks. A virtual reality software system, the MR toolkit, developed in the University of Alberta, provides the low level software support for these devices [51]. They include data sampling and filtering procedures. Research results on how to efficiently navigate a 3D environment are also available for us to use [13] [52] [57].

The obtained paths need to be sampled and recorded so they can be used repeatedly to test all multi-resolution algorithms. In our framework, both level 1 and level

2 paths are recorded as a sequence of 3D positions and quaternions. This format is easy for recording, and sampling 3D translations and rotations.

A real-time rendering benchmark is generated by applying a navigation path to a virtual environment.

6.3 Benchmark parameters

After solving the above technical issues of real-time rendering benchmarks, a benchmark can be easily composed using the virtual environment described in a VRML file and a navigation path. They can be loaded into our testbed and used to measure multi-resolution algorithms.

In a performance study, the benchmark characteristics that affect the performance of test subjects are called *parameters*. Providing a complete list performance parameters in a performance evaluation system is important for performance comparison and analysis. In our performance evaluation framework, the parameters are classified as system parameters and task parameters. System parameters are the parameters of the testbed itself, which include the parameters of support hardware and software platforms. Since algorithms are measured and compared on the same platform, these parameters do not vary among various algorithms. The system parameters include:

- CPU speed/status
- Memory
- Graphics card
- Network status
- Operating System type
- Single User/Multiple User
- 3D input devices
- Display devices
- Compiler

- Graphics API

Task parameters are characteristics of real-time rendering benchmarks. They could vary from one test to the next. These parameters tend to affect the algorithm performance more than other parameters. They are often chosen for performance analysis and comparison. In real-time rendering measurements, task parameters include

- The virtual environments, including geometric objects and their attributes, transformation/group hierarchy and lighting.
- Navigation paths. Level 1 paths include key points, interpolation methods, and number of samples. Level 2 paths include sample frequency, number of samples and 6DOF tracker control method.
- Number of critical vertices

When reporting an algorithm's performance, both system and task parameters should be included. For complete performance analysis, tasks parameters should be studied.

6.4 Chapter summary

In this chapter, a real-time rendering testbed for multi-resolution algorithms is presented, in which various 3D navigation tasks can be loaded and run to measure a range of algorithms. The testbed is based on a standard scene graph and navigation paths. VRML 2.0 files and paths can be automatically loaded and used to generate various real-time rendering benchmarks. It avoids the tedious work of building these tasks. It also provides a unified interface for plugging in diverse multi-resolution algorithms.

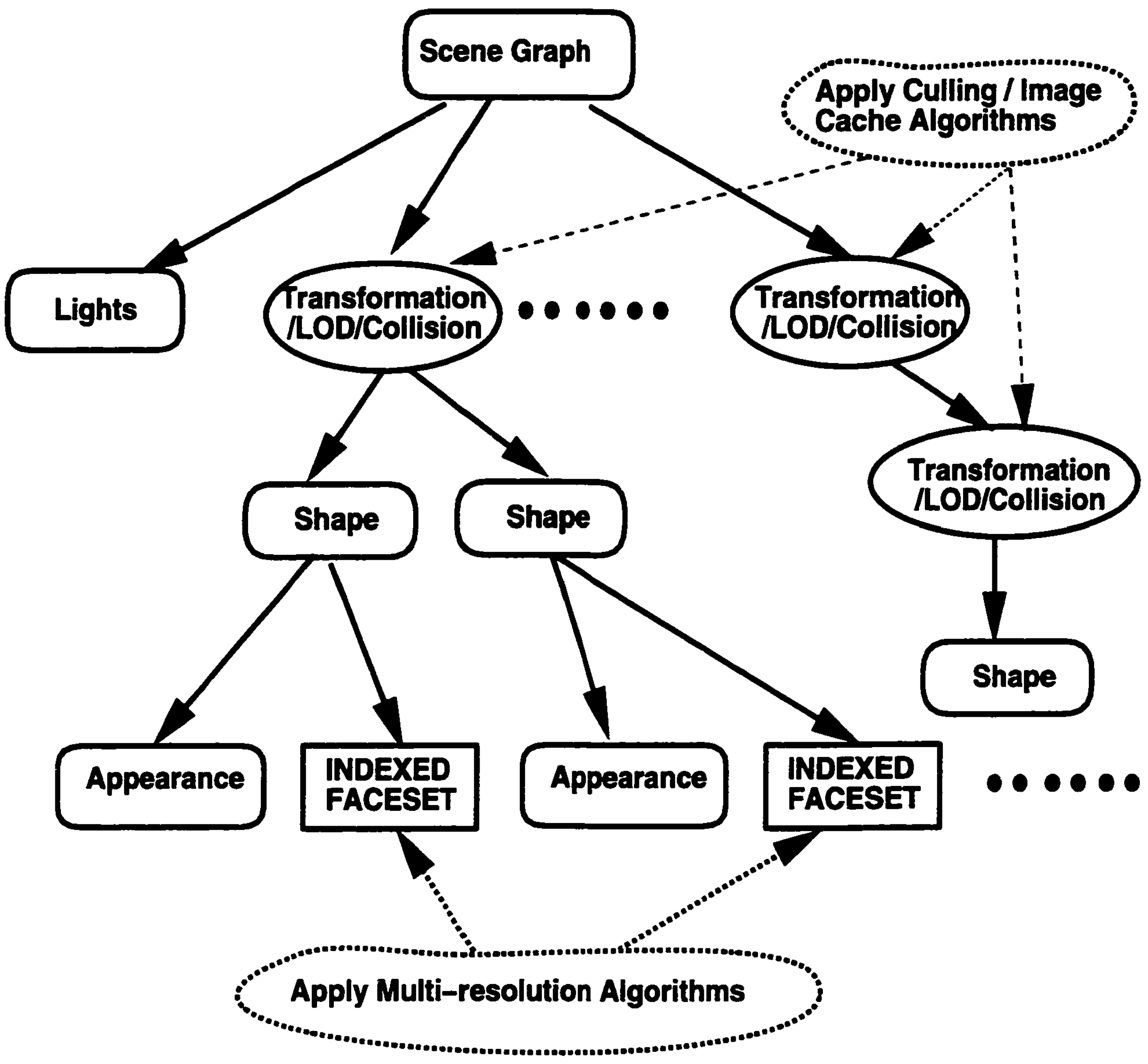

Scene Graph
Apply Culling / Image Cache Algorithms
Lights
Transformation /LOD/Collision
● ● ● ● ● ●
Transformation /LOD/Collision
Transformation /LOD/Collision
Shape
Shape
Shape
Appearance
INDEXED FACESET
Appearance
INDEXED FACESET
● ● ● ● ● ●
Apply Multi-resolution Algorithms

Figure 6.4: Scene graph architecture

Chapter 7

RRB — A Prototype System

To study the merits of our ideas, and use them to evaluate current multi-resolution algorithms, a prototype system, called "RRB", has been developed. It integrates the automatic measures and the real-time rendering testbed and provides a standard system environment for automatically evaluating multi-resolution algorithms. In this chapter, its implementation is briefly discussed. Details are provided in appendix A. Some of RRB functionality is also described in this chapter, demonstrating the potential easy of use and flexibility of the performance evaluation framework.

7.1 Implementation overview

The implementation started in early 1997. Little is left from the first version as numerous revisions have been made, reflecting the insights gained in investigating the performance evaluation problem of multi-resolution algorithms.

RRB realizes the framework system architecture and integrates the techniques of automatic measurement and real-time rendering testbed design that were described in the preceding chapters. It is intended to simplify the tedious effort of performance evaluation, and provide a standard environment to produce consistent performance results which later can be used for performance comparison.

As an object-oriented application framework, RRB consists of a collection of related classes. Figure 7.1 shows the main classes that form the architecture of RRB.

Each box represents one class with its main functionalities. The lines between the classes show collaboration relationships. From the figure we can see that the class of RRBWorkspace connects to almost all the other classes. It is the center of

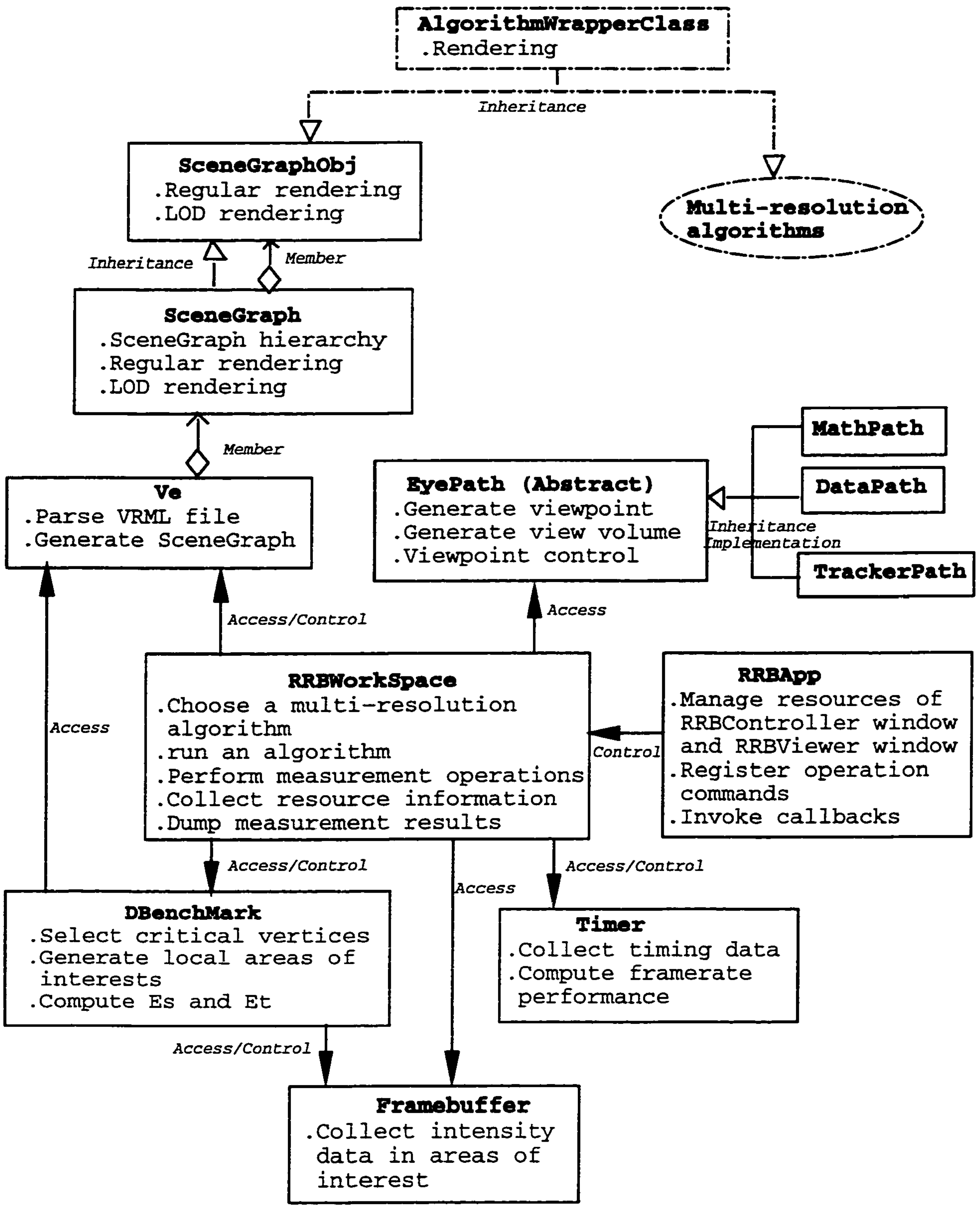


Figure 7.1: RRB architecture and class relationship

the system structure. The figure also demonstrates how a multi-resolution algorithm interacts with RRB. By adding a wrapper class, an algorithm can interface with RRB without changing its internal implementation. Thus it is easily plugged into RRB for performance evaluation. In the next section, an example is used to show the interfacing technique.

7.2 Interface between multi-resolution algorithms and RRB

RRB is a performance evaluation framework for multi-resolution algorithms. At the moment, it supports the regular rendering algorithm and level of detail algorithms based on the distance between the viewer and objects. Level of detail algorithms can be evaluated directly using RRB, providing the LOD models and distance criteria. Other multi-resolution algorithms need to be plugged into RRB for measurement.

As discussed in chapter 6, most multi-resolution algorithms work on the geometric node of IndexedFaceSet. Some are combined with culling techniques or hierarchical cache to improve rendering speed. In either case, RRB provides an interface to plug in the algorithm, which is actually a derived class of SceneGraphObj. The SceneGraphObj class is a base class which defines the basic interface for all scene graph objects. The set of scene graph node classes implemented in RRB are:

- Appearance
- Box
- Collision
- Color
- Coordinate
- DirectionalLight
- Group
- ImageTexture

- IndexedFaceSet
- LOD
- Material
- Normal
- PointLight
- Shape
- SpotLight
- TextureCoordinate
- TextureTransform
- Transform

Each of the classes maintain a set of scene graph attributes and provide the rendering methods. They together form the scene graph hierarchy and implement the regular rendering algorithm of a virtual environment.

Among them, IndexedFaceSet is of interest to most algorithms. Here, Lau's simplification list [34] is used as an example to show how a typical multi-resolution algorithm is interfaced with the IndexedFaceSet class in RRB. The algorithm consists of eight classes. Only one that provides the final simplification and recovery methods is interfaced with RRB. The rest are treated as black-boxes. To interface the algorithm with RRB, a wrapper class is needed to connect them together. Internal implementations of both RRB and the algorithm are not changed. Figure 7.2 shows the class diagram of the interface.

It is the framework system architecture that enables the clear and simple hookup. Complex real-time rendering system configuration and real-time rendering benchmarks are all handled in RRB itself. After the hookup, RRB can load and run various real-time rendering benchmarks and produce the performance data of the algorithm.

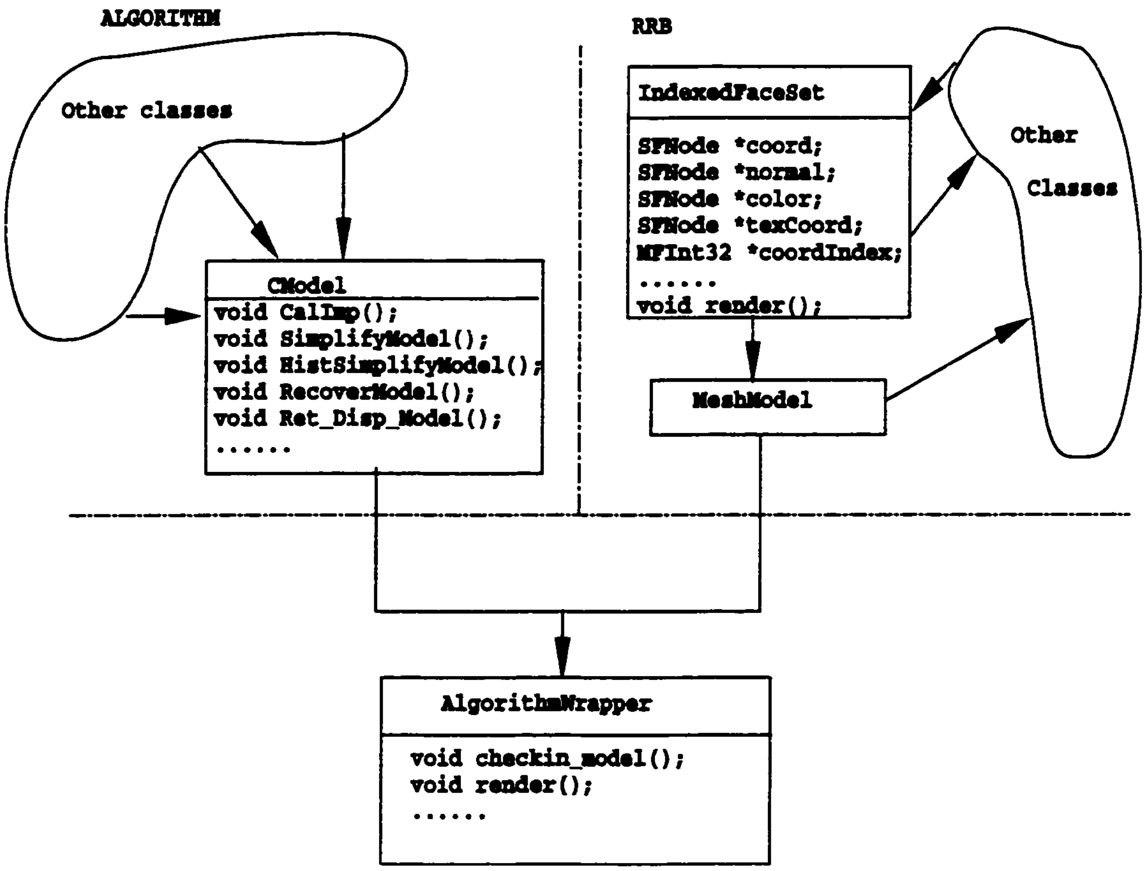


Figure 7.2: Interfacing the algorithm with RRB

7.3 Performance evaluation with RRB

For users' convenience, RRB provides a 2D GUI to facilitate measurement operations. It has a control window which facilitates measurement operations, and a view window providing a visual display of how an algorithm runs the real-time rendering benchmarks. Figure 7.3 shows a snapshot of RRB.

Using RRB, the measurement of a multi-resolution algorithm can be done in just a few steps.

The first step is to load a real-time rendering benchmark. A real-time rendering benchmark consists of two components: a virtual environment and a navigation path. A user first loads a virtual environment from a VRML 2.0 file. After the user specifies the file, RRB is responsible for parsing the file and generating the scene graph. Then, the user can specify the navigation path, which can either be a level 1 path or a level 2 path. A default path is defined in the system, if she chooses not to select one by herself. RRB will automatically map the path on the loaded virtual environment and display frame 0. The loaded benchmark can be used to test multiple algorithms.

The second step is to measure the chosen algorithm's performance for this benchmark. An algorithm's frame rate, pre-processing time and resource consumption can be obtained while it runs the benchmark. The user initializes the algorithm by clicking the corresponding radio button in the control window. The measurement procedure is then activated by just clicking the *Navigation* button. RRB records timing data of the pre-processing procedure and the rendering of each frame. Once the task is over, the user can click *FrameRate* to dump the frame rate, algorithm pre-processing time, and memory usage, along with current platform related information to a data file. Measurement of the algorithm's image fidelity is also done by clicking a few buttons. First, by clicking *OrgInfo*, the user instructs RRB to automatically choose critical vertices and obtain the positions and intensity of areas of interest produced by a regular algorithm for this benchmark. This information is kept in RRB for repeatedly measuring multiple algorithms until a new virtual environment or path is loaded. Then, the algorithm runs the benchmark, and its spatial and temporal image fidelity are computed once button *EsEt* is clicked. Button *ImgQ* is for dumping the image fidelity results.

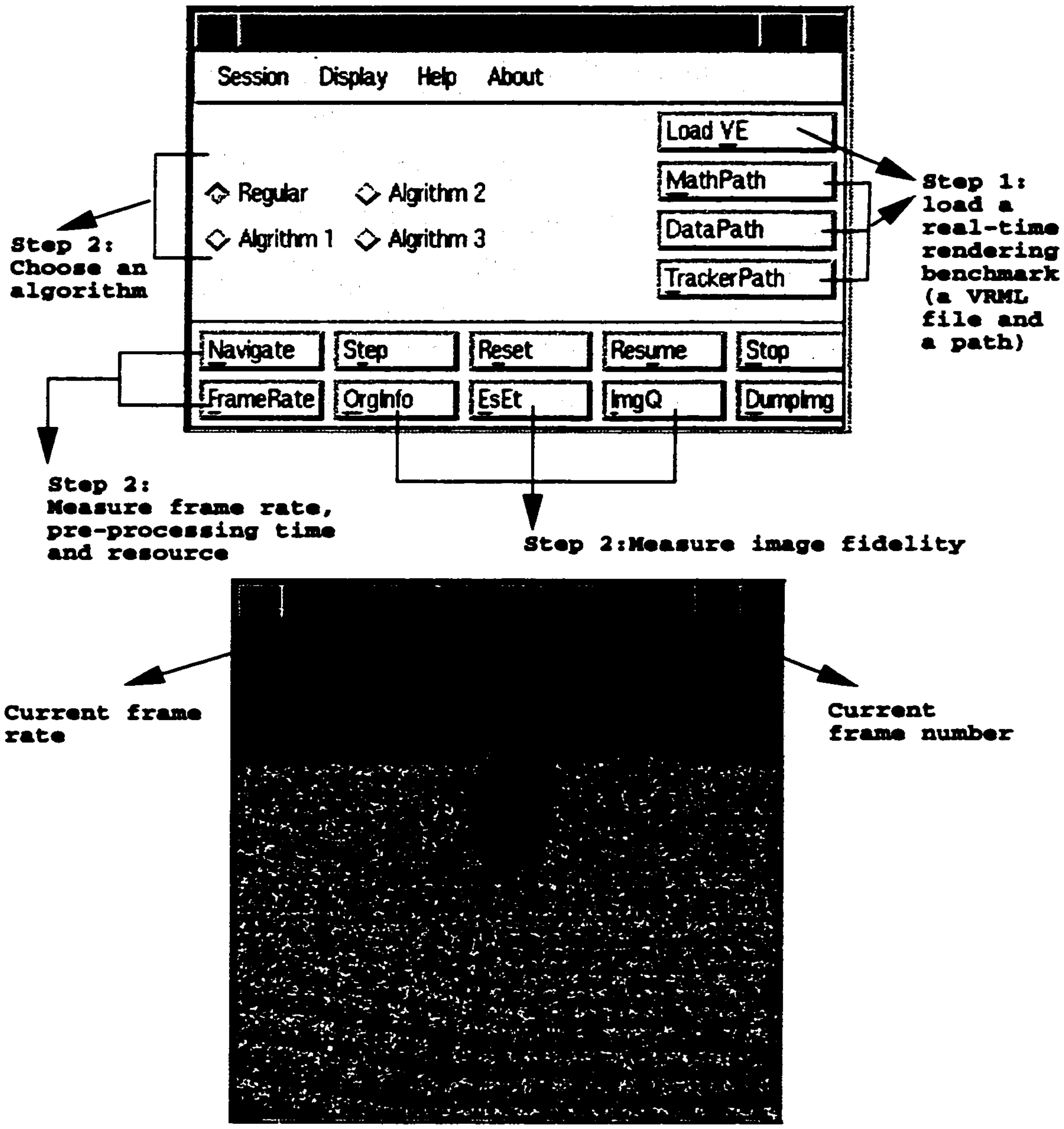


Figure 7.3: RRB snapshots

In RRB, all the GUI components for measurement operations are activated or deactivated automatically in a constrained sequence to help users reduce errors. While far from a complete set of performance measurement tools, RRB provides a reasonable base for most performance evaluation studies of multi-resolution algorithms that we discuss in this document.

7.4 Chapter summary

In this chapter, we introduce the prototype system RRB, which realizes the performance evaluation techniques discussed in the preceding chapters. It is shown that performance evaluation of a multi-resolution algorithm with RRB is easy. RRB avoids the effort of real-time rendering system configuration and benchmark development. Thus, it significantly reduces the work of performance measurement.

Chapter 8

Evaluating the Framework

In previous chapters, the fundamental ideas of the performance evaluation framework of multi-resolution algorithms and the prototype system were described. This chapter demonstrates that RRB is capable of automatically measuring various multi-resolution algorithms and producing consistent performance data.

8.1 Evaluation method

The main purpose of this research is to investigate ways of automatically evaluating multi-resolution algorithms. Since RRB is the first performance evaluation framework, there is no similar existing system to compare with. One way to verify the results is to demonstrate that the performance rating generated by our RRB are in accordance to the expected ratings of the algorithms, and also RRB indeed avoids the tedious effort of complex system configuration and real-time rendering benchmarks.

It is very difficult, if not impossible, to obtain the actual performance ratings of various algorithms at the moment, because no performance evaluations and comparison have been done. This was the main reason for developing this framework. One practical way to break the deadlock situation and proceed with evaluating RRB is to artificially produce some algorithms, whose ratings are known or can be predicated. Then, compare the ratings generated by RRB with the known ratings. If the two results match, then the effectiveness of RRB is evident.

Such algorithms can be generated from one basic algorithm by changing key strategies and parameters. We choose Lau's continuous multi-resolution algorithm [34] as the base algorithm. It is a typical continuous multi-resolution algorithm. The algorithm simplifies an original mesh using edge collapse strategies and keeps the simplification steps in memory. Later at run time, the simplification list is reused to reduce or increase the level of detail adaptively. As discussed in chapter 3, many recent multi-resolution algorithms share similar ideas, such as Hoppe's progressive mesh and Xia's merge tree. We use Lau's algorithm as a basis for producing algorithms that can easily be plugged into RRB and measured in the standard real-time rendering environment. The other advantage of choosing a continuous multi-resolution algorithm, such as Lau's simplification list, as the base algorithm is that it can also be used to directly produce fixed level of detail models, and thus generate LOD algorithms. By changing the runtime display mechanism, other multi-resolution algorithm variants can be generated from it as well.

To verify the result of RRB, Lau's algorithm itself, an LOD algorithm, two multi-resolution algorithms derived from Lau's algorithm and the regular rendering algorithms are used.

- *Algorithm Rynson* is Lau's simplification list. It is designed to maintain a target frame rate and also preserve the image fidelity as much as possible. The algorithm computes the visual importance of each vertex and edge in a preprocessing procedure. The lower resolution model is obtained by collapsing edges in the original model starting with those in the lowest importance group. The edge collapsing operations are cached in a simplification list. During rendering, this cached simplification list is used to adaptively reduces or increases level of detail. Algorithm Rynson uses Funkhouser's predictive optimization method to select levels of detail in real-time rendering. It predicts the complexity and visual importance of objects from the current viewpoint and chooses an appropriate level of detail for each object to meet the target frame rate. The frame rate produced by Algorithm Rynson is expected to be the best among the algorithms. Its image fidelity is also expected to be very good. However, the preprocessing time and memory cost are high due to producing and maintaining the cached simplification list.

- *Algorithm LOD* is a typical level of detail algorithm. It does not maintain a continuous multi-resolution model in memory, like Rynson. Instead, it uses two pre-generated level of detail models and the original model itself. The two models have 50% and 25% faces of the original model respectively. During rendering, it switches level of detail according to the distance between the viewer and the model. Therefore, it does not produce guaranteed frame rate. Its image fidelity could vary on different real-time rendering benchmarks. When the distance between the viewer and an object change very much and very often in a benchmark, the algorithm is expected to produce high temporal image distortion, which are popping artifacts; otherwise it does not. Its spatial image fidelity is expected to be okay. The algorithm does not have extra pre-processing time online. However, the level of detail models have to be pre-generated off-line. Its extra memory costs are mainly for storing the coarser level of detail models.

- *Algorithm XPop* is designed to produce poor frame rate performance and image fidelity performance. It uses a feedback mechanism for level of detail control. The original model is rendered in the first frame. As the frame rate is very low, the algorithm traverses the simplification list and renders a very coarse level of detail which has only 10% of the faces of the original model in the second frame. Then, in the third frame, it recovers to the original model from the coarse model using the simplification list as the feedback from the second frame shows the algorithm can render a high level of detail. Thus, the XPop algorithm is designed to switch between the original model and a very coarse model in every frame transition. It is expected to have a large temporal image distortion, as it switches from the finest resolution directly to the coarsest back and forth. Its spatial image fidelity may not be the worst, since half of the frames use the original models. The algorithm's average frame rate should not be very bad because it does not need to spend time to do online simplification with the cached simplification list. Many of the current proposed continuous multi-resolution algorithms, such as progressive mesh and merge tree are expected to benefit from a similar idea and improve frame rate. However, the performance of such an idea has never been tested. No report has been published on how much performance loss can be caused by only traversing the vertex hierarchy.

Algorithm Xpop is included to verify the idea and see if such an idea really always improves rendering speed.

- *Algorithm MPop* uses same method as XPop, except that the coarse level of detail has 25% of the faces of the original model. It is expected to have better image fidelity and frame rate performance than XPop.
- *Algorithm Regular* is the regular rendering algorithm, which renders the original models at all times and does not involve in any multi-resolution strategies. It is included in the experiments to test the boundary conditions for all our measures. Among all the algorithms, it should have the best image fidelity, which means it does not have spatial and temporal distortions. The original algorithm is expected to have the slowest frame rate, however, frame rate does not necessarily fluctuate. For the regular algorithm, the fluctuating frame rate can be produced when the scene complexity varies from one frame to another due to viewpoint changes. It can also be constant most of time, when the scene complexity does not change much during real-time rendering. This original algorithm does not have any extra resource consumption and preprocessing time, compared with the other algorithms.

We choose these algorithms which represent most of the typical approaches used in current multi-resolution algorithms. Their performance are basically known to us. In the next section, we will present the experiments of measuring their performance with RRB and determine if the results produced by RRB match with the expected ratings of the algorithms.

8.2 Measurement results and analysis

All algorithms, except for the regular rendering and the LOD algorithm, need to be plugged into RRB before the experiments. The interface procedure is simple and is described in detail in §7.2. The regular algorithm and LOD algorithm have been integrated in RRB system. Then RRB is responsible for loading specified real-time rendering benchmarks from VRML and path files. Performance data was produced while algorithms perform the benchmarks in RRB. RRB is installed on a Silicon

(a) frame 1 (b) frame 15 (c) frame 30

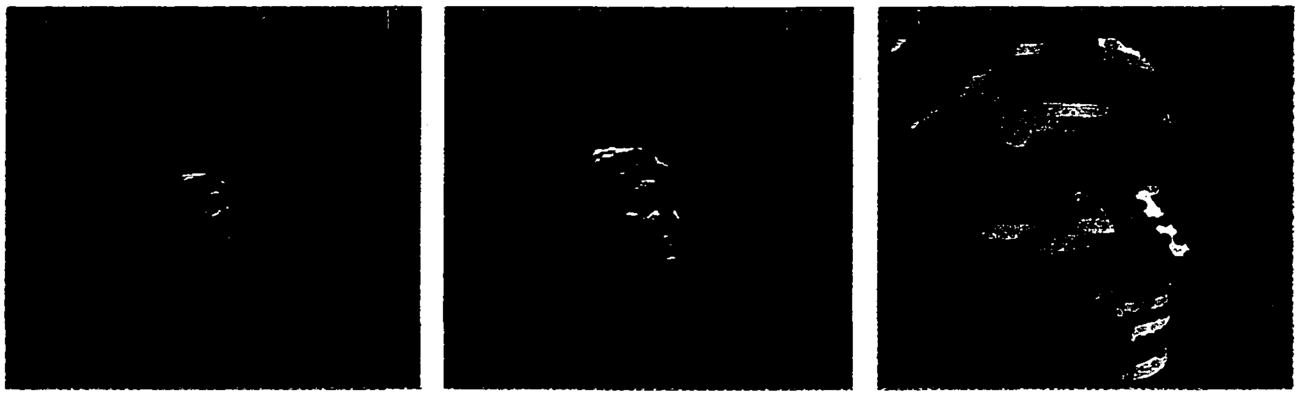
Figure 8.1: Sample frames in experiment 1

Graphics Crimson with RealityEngine, R4400, 150 MHZ CPU and R4000 FP Processor, 64M memory, 21" monitor (1280x1024) at 60Hz, IRIX 6.2, OpenGL, X11, Ansi C++ development and executable environment. The system is on the department network. However it runs in single user status in all the timing experiments, as all remote logins are blocked during these periods of time. The performance data of all five algorithms listed in the following sections are produced in this standard environment. All the experiments and measurement results are repeatable with RRB.

8.2.1 Experiment 1

Experiment goals and setup

The first experiment is to test if RRB can load a simple real-time rendering benchmark, run the five algorithms on it and measure their performance results in terms of frame rate, Es, Et, preprocessing time and resource consumption. The benchmark is a typical real-time rendering task, a viewer walking towards and then away from a "face" model. The required frame rate is 20 frames per second, i.e. 50 msec per frame. All the parameters of the benchmark are described in a VRML file, head.wrl and a path file, ex1. Figure 8.1 and 8.3 show three sample frames produced by the regular algorithm and the path. There are 60 frames in the benchmark in total. 391 critical vertices, including 193 extrema of curvature and 198 discontinuities, are used in Es and Et. Figure 8.2 illustrates the area of interests projected by critical vertices in the sample frames.



(a) frame 1 (b) frame 15 (c) frame 30

Figure 8.2: Areas of interest in sample frames in experiment 1

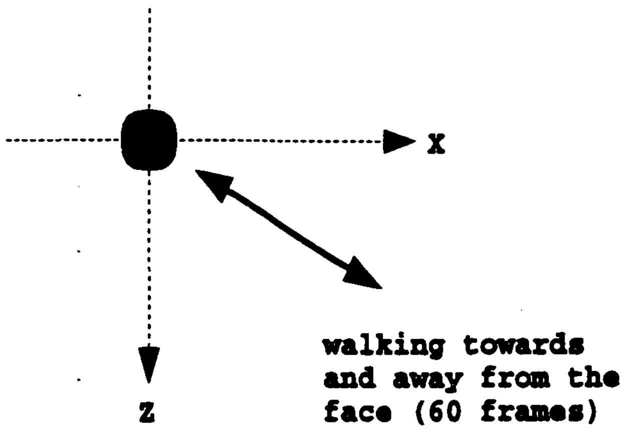


Figure 8.3: Navigation path in experiment 1

Algorithm	$\bar{f}$ (msec)	$minf$ (msec)	$maxf$ (msec)	F	Es	Et
Rynson	46.433	45	50	3.895556	1.00367	0.403666
LOD	52.6	37	164	19.0733	3.61027	1.35122
XPop	684.783	223	1150	460.35	6.68442	6.42103
MPop	564.233	194	939	369.1.914	2.129522	2.36024
Regular	145.383	145	156	0.613332	0	0

Algorithm	Preprocessing time (msec)	Process Memory (kb)
Rynson	35965	11196
LOD	Manual	6068
XPop	36503	11252
MPop	36358	11264
Regular	0	5632

Table 8.1: Performance results in experiment 1

Experiment results and analysis

Table 8.1 lists the performance data of the five algorithms produced by RRB. They include their average frame time, minimum frame time, maximum frame time, fluctuation of frame time, Es and Et. The results clearly indicate the ratings of the five algorithms. Rynson has the best frame rate performance, in terms of both threshold and consistency. To maintain the required frame rate, it varies the resolution from 66% – 69% of original faces. It thus has the best spatial image fidelity. It has a little temporal image distortion. This is mainly because the algorithm changes the resolution adaptively when it has more time for rendering. Rynson uses about 36 seconds pre-processing time to do first round simplification and generate the simplification list. It is relatively high for a model with only 4356 faces. The memory cost is 11196kb, which is also high compared with 5632kb used in the original algorithm.

LOD improves the frame rate, however, it does not guarantee consistent frame rate performance, since the algorithm itself changes the resolution only based on the distance between object and viewpoint. When the object is far, it uses the coarsest level of detail which requires 37 msec for rendering. When it is close to viewpoint, it switches to the fine resolution model, and spends 164 msec, first on making the decision and then on rendering. During rendering, the viewer experiences navigation at different speeds which is out of her control. Its overall spatial image fidelity ranks after Rynson and MPop, because it has to use lower levels of detail most of the time, but it is better than XPop.

XPop is one of the worst examples of using simplification list in real-time rendering. It does not improve frame rate at all, and even produces slower and more variable frame rates than the regular algorithm. This bad frame rate performance of XPop indicates that traversing the simplification list only can be expensive. The cost of 36 seconds pre-processing time and about twice the amount of memory doesn't pay off because of improper use. Therefore, it is evident that a vertex hierarchy, such as a simplification list, progressive mesh, or merge tree, may not always be effective for speeding up rendering as commonly believed. Their performance must be evaluated thoroughly. The spatial image fidelity of XPop is also the worst among the five algorithms, because it renders the model with 10% faces in half of all frames. XPop has more severe temporal image distortions, as expected. It switches between the original model and the coarse model at every frame transition.

The frame rate and image fidelity performance of MPop is better than Xpop as expected. Its spatial image fidelity is even better than LOD, because half of all frames use original models, while LOD uses coarser level of details in most of the frames in this benchmark.

The regular algorithm produces slow but constant frame rate in this benchmark, because the scene complexity does not vary much. It does not produce spatial and temporal errors as expected.

Overall, the performance data produced by RRB basically matches with the expected ratings of the algorithms in this experiment. Since the data are all produced in the same environment – RRB, they can be used to fairly characterize and compare these algorithms' performance.

8.2.2 Experiment 2

Experiment goals and setup

Algorithms tend to behave differently on different tasks. In this experiment, the navigation path is changed to circling around the "face", as shown in figure 8.4. The distance between the viewer and the object changes less often than in the first experiment. Also, in most of the frames, the viewer is closer to the object than in

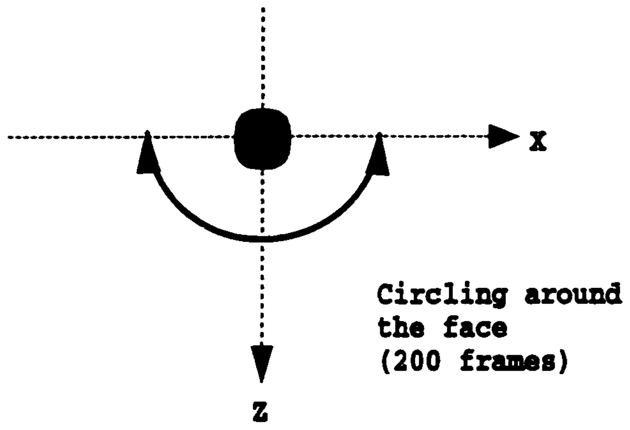


Figure 8.4: Navigation path in experiment 2

experiment 1. All the other parameters remain the same as in experiment 1. This change is expected to affect the LOD algorithm the most. Since it selects level of details based on distance. When distance remains relatively short and constant, LOD uses the models of fine resolutions. It also does not need to switch the level of detail often. All the other algorithms should not change much, since we still use the same model and thus the frame rate should not vary too much.

Experiment results and analysis

Tabie 8.2 shows the performance results produced by RRB. By comparing the performance data in the two experiments, we can see that the performance of LOD in benchmark 2 does differ a lot from benchmark 1. The image fidelity improves a lot, which is even a little better than Rynson. The result is expected since LOD renders the original model in most of its frames. It also produces much slower frame rate. LOD also produces less temporal image distortions than in benchmark 1 due to the navigation path change. The other algorithms' performance vary very little from benchmark 1 as expected. In this experiment, RRB captures the performance changes of the algorithms. It shows that LOD performs better in benchmark 2 than in benchmark 1. Algorithm Rynson performs very good in both benchmarks. Its performance is affected little by the change of navigation path.

Algorithm	$\bar{f}$ (msec)	$minf$ (msec)	$maxf$ (msec)	F	Es	Et
Rynson	45.64	45	48	0.672	1.1277	0.36799
LOD	103.58	72	156	34.5608	1.05318	0.38057
XPop	686.03	223	1168	461.59	6.48925	5.50483
MPop	568.31	195	1066	372	2.41492	2.16531
Regular	146.27	145	150	0.439303	0	0

Algorithm	Preprocessing time (msec)	Process Memory (kb)
Rynson	36379	11216
LOD	Manual	6096
XPop	36886	11268
MPop	36546	11276
Regular	0	5632

Table 8.2: Performance results in experiment 2

8.2.3 Experiment 3

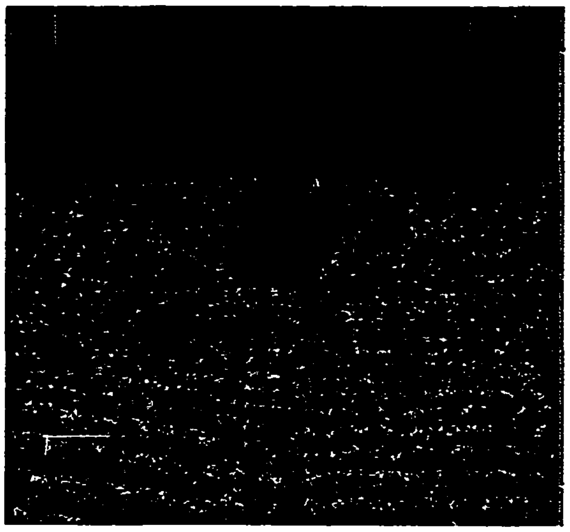
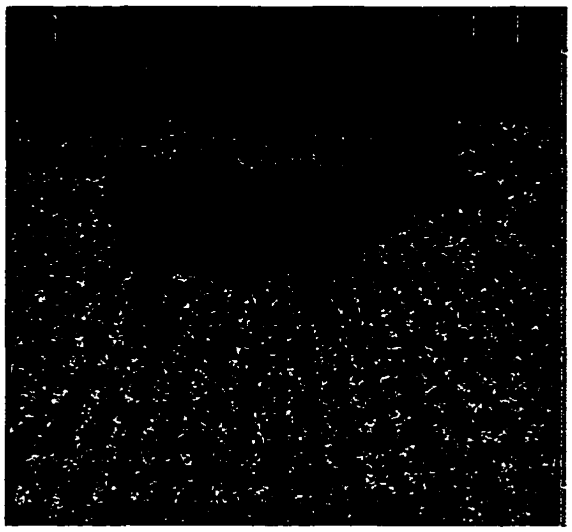
Experiment goals and setup

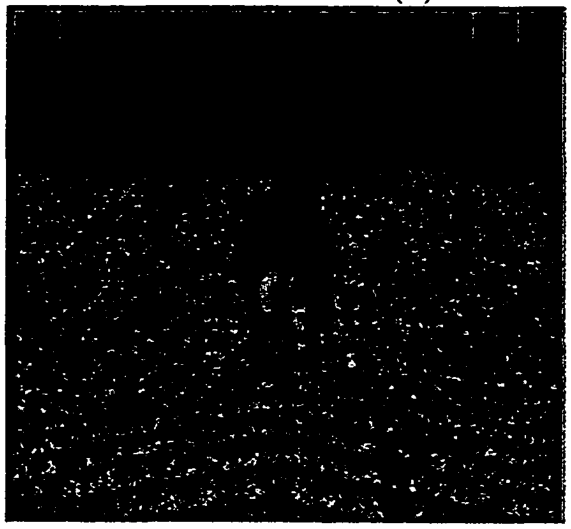
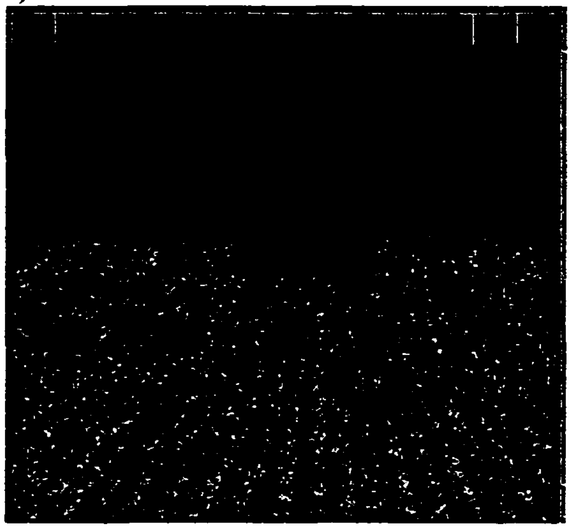
The first two test benchmarks are quite simple. In order to test if RRB can load more complex real-time rendering tasks, apply the algorithms on them and automatically measure their performance, we designed a third experiment. The real-time rendering benchmark consists of a more complex scene and a longer path. The scene is composed of five indexed face sets with different attributes. Together they describe a colored cow on grass land. The path is walking on the grass and looking at the cow. All the parameters of the benchmark are described in the VRML file, myworld.wrl and the path file, ex3. Figure 8.5 and 8.7 show the sample frames produced by the regular algorithm and the path.

In the benchmark, 958 critical vertices, including 594 extrema of curvature and 364 discontinuities, are used in Es and Et. Figure 8.6 illustrates the areas of interest projected by critical vertices in the sample frames.

Experiment results and analysis

Since RRB provides a standard scene graph hierarchy, applying a continuous multi-resolution algorithm, such as Rynson, MPop, and XPop, on the complex benchmark is the same as applying the algorithm on a simple benchmark with only one indexed face set. By default, RRB automatically traverses the scene graph from top to bottom



(a) frame 1 (b) frame 30



(c) frame 59 (d) frame 90

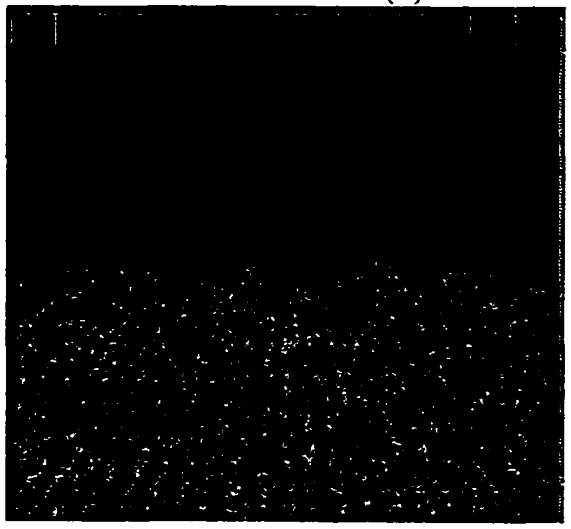
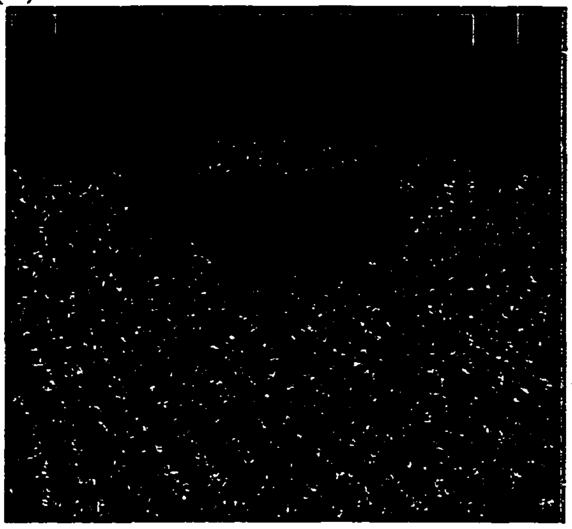
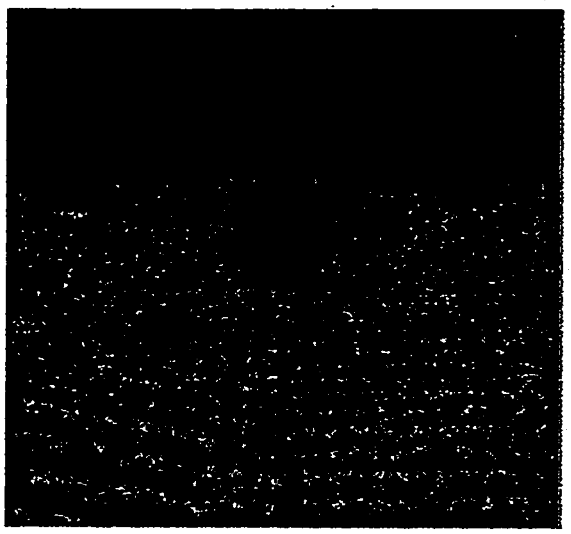
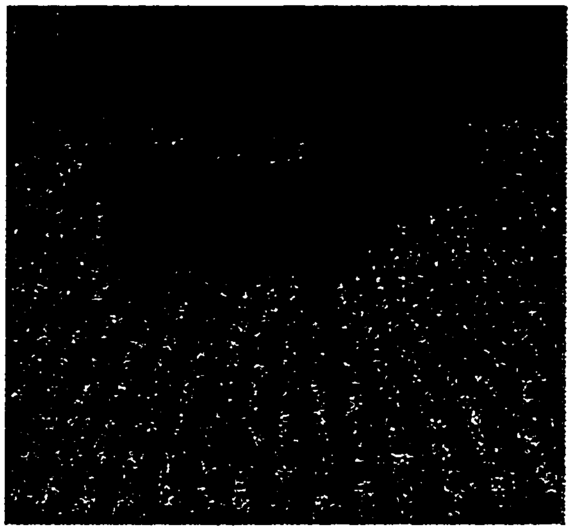
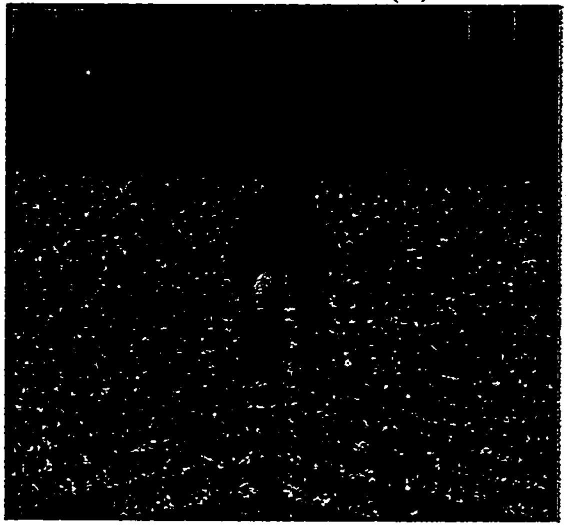
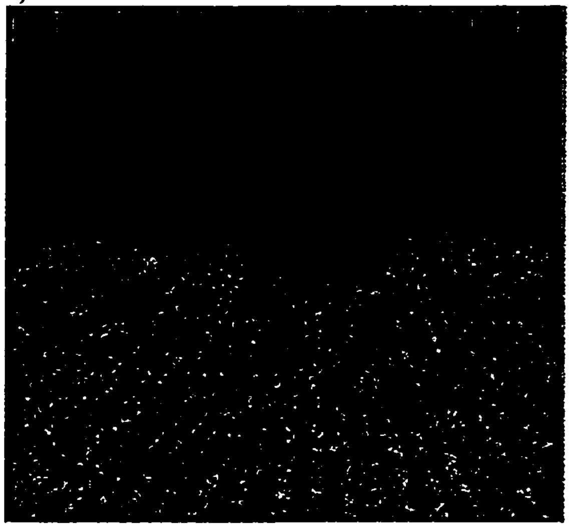


(e) frame 108 (f) frame 132

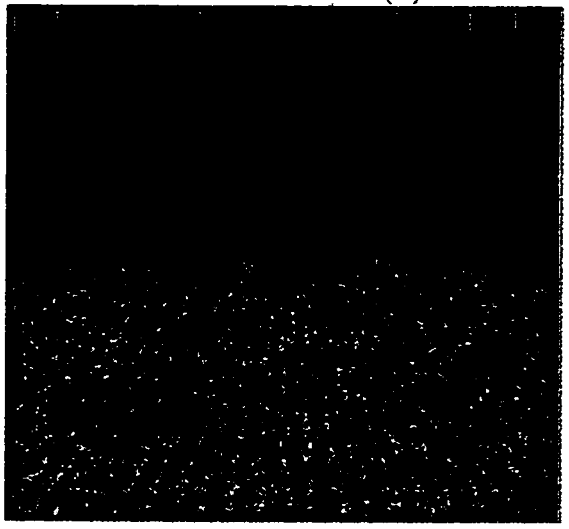
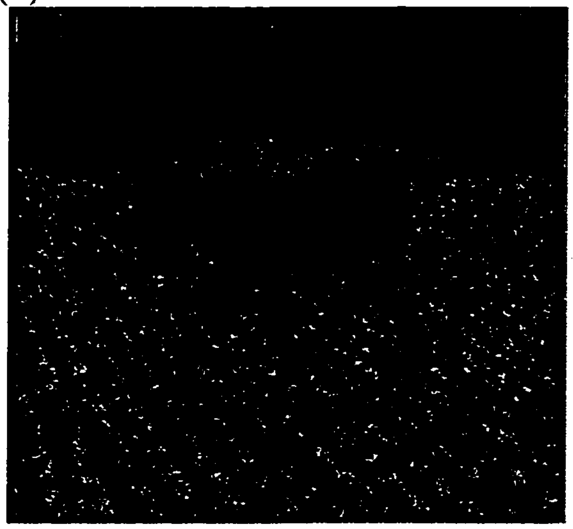
Figure 8.5: Sample frames in experiment 3

(a) frame 1 (b) frame 30

(c) frame 59 (d) frame 90

(e) frame 108 (f) frame 132

Figure 8.6: Areas of interest in sample frames in experiment 3

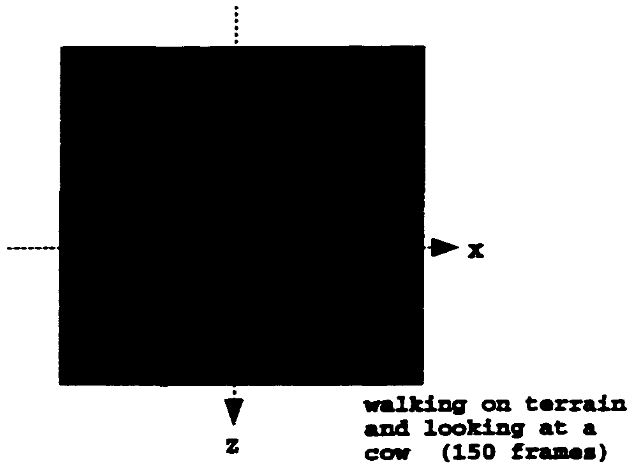


Figure 8.7: Navigation path in experiment 3

and left to right, and applies the algorithm on each of the indexed face sets one by one. Different scene graph traversing and culling techniques can certainly be added on as well, with the flexible transformation hierarchy of the RRB scene graph. Therefore, the user does not need to do extra work for measuring these algorithms on benchmarks of various types. However, for LOD algorithms, the level of detail models for each of the indexed face sets in the virtual environment have to be generated and re-composed manually. For simple virtual environment with only one surface model, such as the ones in the first two benchmarks, it does not take too much time. If we have a complex scene, the LOD models have to be generated and re-composed one by one. At the moment, it is found to be the most time consuming work for performance evaluation using RRB. In the case of benchmark 3, we spent more than three hours to put together the LOD models of five surface models in VRML format. This situation is expected to improve with a LOD modeler and composer for VRML.

Nevertheless, RRB successfully measures the algorithms on the benchmark and produces the performance data, as shown in table 8.3. Due to increased scene complexity, all algorithms except Rynson, produce slower frame rates than in the first two benchmarks. Their frame rate rankings remain the same. Algorithm Rynson still ranks the best among all the algorithms in terms of frame rate and image fidelity. However, its spatial image fidelity is not much better than other algorithms because it has to trade off more image quality to maintain the required frame rate. All the algorithms need more resources due to the increased scene complexity. Also, algo-

Algorithm	$\bar{f}$ (msec)	$minf$ (msec)	$maxf$ (msec)	F	Es	Et
Rynson	50.52	49	54	1.02934	1.61971	0.977169
LOD	86.34	52	223	34.9848	2.02177	1.11837
XPop	861.92	290	1469	571.147	1.85416	1.54629
MPop	705.973	249	1220	456.16	1.66947	1.20789
Regular	200.493	199	213	1.75004	0	0

Algorithm	Preprocessing time (msec)	Process Memory (kb)
Rynson	43025	14100
LOD	Manual	7080
XPop	43075	14192
MPop	42048	14192
Regular	0	6400

Table 8.3: Performance results in experiment 3

rithm Rynson, MPop and XPop need more pre-processing time. As we can see, all the results produced by RRB are in accordance with the expected ratings.

8.2.4 Section summary

As we have shown in the experiments, RRB is capable of measuring the five algorithms and producing consistent measurement results. The results quantitatively show which algorithm is better than others. They are in accordance with the expected rankings of the algorithms. RRB also makes it possible to compare an algorithm's performance in benchmarks of different scale and complexity, and helps the user to understand under what conditions an algorithm performs the best.

8.3 Important parameter analysis

As we have discussed in §6.3, benchmark parameters affect the measurement results of algorithms. This fact is also illustrated in our experiments in §8.2. Among the many parameters, virtual environment, navigation path, dimension of viewing window and distribution, and number of critical vertices are the most important parameters. It would be very interesting and useful to further study their effects in performance analysis of multi-resolution algorithms in real-time rendering. However, due to the limitation of resource and time, we only conducted some experiments on the number of critical vertices to study their effects on the measurement results of the algorithms'

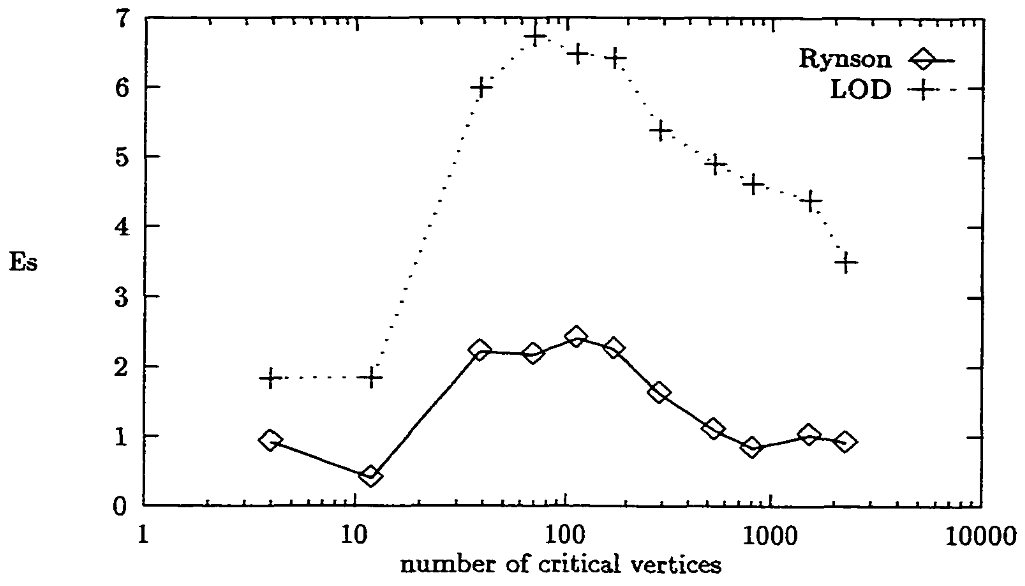


Figure 8.8: Es affected by number of critical vertices in benchmark 1

image fidelity performance.

8.3.1 Number of critical vertices

Previous research suggests that image distortions are more noticeable in extrema of curvature and discontinuities. Thus we choose them as critical vertices and measure the image distortions in these local areas of interest. However, to the best of our knowledge, there is no report on how many vertices are sufficient to effectively capture the noticeable distortions. In the following experiments, we try to determine the answer and find out how the number of extrema of curvature affects Es and Et.

First, we choose algorithm Rynson and LOD to run benchmark 1. Both algorithms do not have dramatically large image distortions. Rynson has a better image fidelity performance than LOD in both spatial domain and temporal domain in benchmark 1. However, it is not much better. These conditions are ideal for us to study the effects of critical vertex number. In this experiment, the number of extrema of curvature is increased from 4 to 2278 (all vertices). Discontinuities (198 vertices) are not included in the experiments, except for the last case in which all vertices are included. Figure 8.8 and 8.9 illustrate the results.

As shown in the graphs, the number of critical vertices affects the measured results. The detected image distortions are smaller when too few or too many vertices are

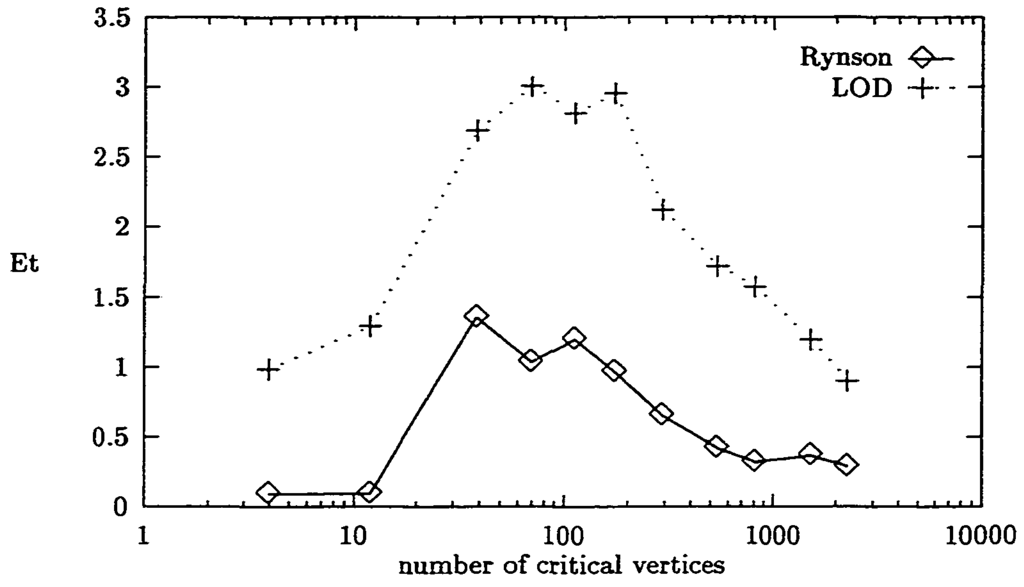


Figure 8.9: Et affected by number of critical vertices in benchmark 1

chosen. The measures perform best when the critical vertices are in the range of $2\% - 20\%$. They produce the maximum absolute errors for both algorithms and also the biggest differences between the two algorithm. It indicates that too few local areas of interest can't represent all the artifacts, while too many even out the distortions.

To test if the same trend applies to other situations, another set of experiments was done. In this second set of experiments, the two algorithms are tested using benchmark 3. Benchmark 3 has a complex scene, which has five meshes. Each of them uses different color or texture. The discontinuities are considered more important in this benchmark than in benchmark 1. Thus, they (364 vertices) are always chosen as critical vertices in this set of experiments. The number of extrema of curvature increases from 12 to 3186 (all vertices). Figure 8.10 and 8.11 illustrate the measurement results.

The measurement results demonstrate a similar trend. The detected absolute image distortions are relatively small when either too few or too many critical vertices are chosen. The measures perform the best when the critical vertices reach 30% of all vertices. It is obvious that the measures need different numbers of critical vertices to reach their optimal condition in different benchmarks. They need more critical vertices in benchmark 3 than in benchmark 1.

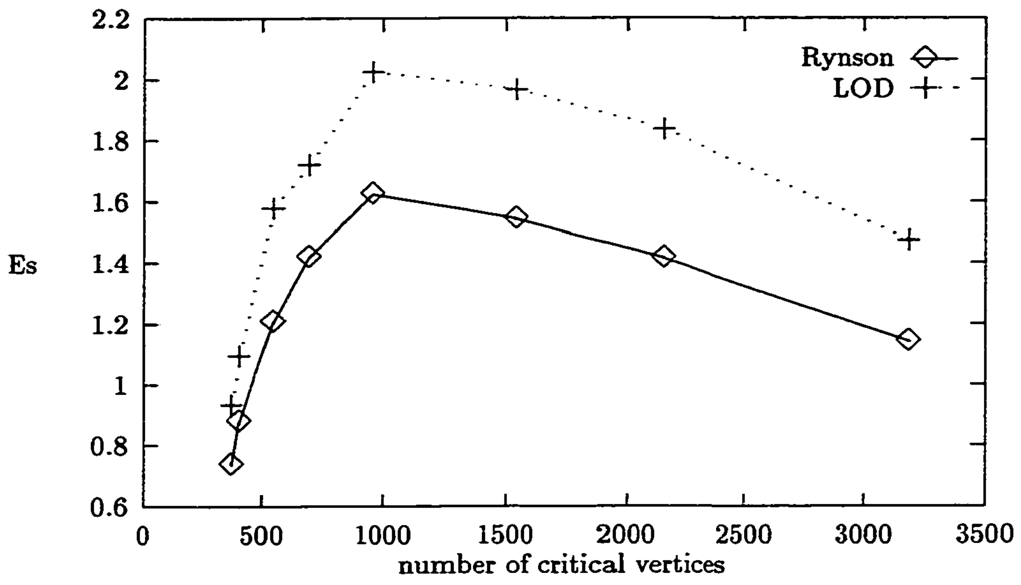


Figure 8.10: Es affected by number of critical vertices in benchmark 3

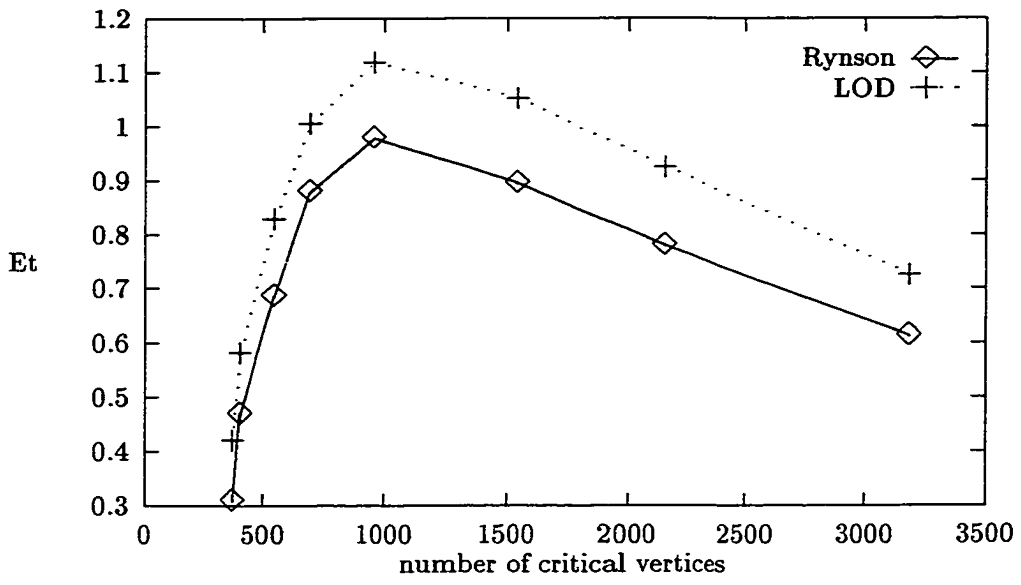


Figure 8.11: Et affected by number of critical vertices in benchmark 3

8.4 Discussion

As we have shown in the experiments, RRB provides a standard real-time rendering environment to measure multi-resolution algorithms. The measurement results produced by RRB are in accordance with the actual ratings of the algorithms. RRB also effectively characterizes the algorithms and detects their artifacts in benchmarks of various types and scale.

In the experiments, some intrinsic limitations of the image fidelity measures are also revealed. They are affected by the benchmark parameters, such as virtual environment lighting, navigation path and the number of critical vertices. Due to time and resource limitation, only formal experiments on the number of critical vertices were conducted. Experimental results show that different number of critical vertices produce different image fidelity measurement results. In these experiments, the results never show a conflicting ranking of algorithm performance. However, a certain number of critical vertices is required to reach the optimal condition for the measures.

8.5 Chapter summary

Experiments have been carried out to evaluate the effectiveness of the performance evaluation framework, RRB. First, we evaluate five multi-resolution algorithms with RRB. By comparing the ratings produced by RRB with the actual rankings of the algorithms, RRB is shown to be capable of measuring a range of multi-resolution algorithms and producing consistent and meaningful performance results. Second, we experimentally examine the effects of a performance parameter - number of critical vertices, on the image fidelity measures. Experimental results show that a certain number of critical vertices are needed for image fidelity measures to reach their optimal condition. However, measures do not produce a conflicting ranking by varying the value of the parameter.

Chapter 9

Applications of RRB

As we have discussed, many multi-resolution algorithms have been proposed for real-time rendering. However, their performance has never been evaluated and compared in the application domain. RRB was developed to fill this gap. This chapter presents our experience with evaluating three typical LOD algorithms using RRB. They serve as more evidence to show the potential of our performance evaluation framework. In addition, some of the potential applications which are made possible by the extensibility and flexibility of RRB in particular are also explored.

9.1 Measurement of LOD algorithms using RRB

9.1.1 Introduction to the LOD algorithms

As we discussed earlier, an LOD algorithm, like other multi-resolution algorithms consist of two components, LOD modeling and the model selection mechanism used in rendering. In our experiments, LOD modeling of the three algorithms was done with three mesh simplification algorithms available in the public domain. They are:

- Cluster: It simplifies a polygon model by collapsing multiple points together. This is basically Jarek Rossignac and Paul Borrell's method of simplifying polygon models [47]. The code is provided by Greg Turk, Georgia Institute of Technology.

- Qslim 2.0: It simplifies a mesh model based on quadric error metrics. The algorithm was developed by Michael Garland, Carnegie Mellon University [23].

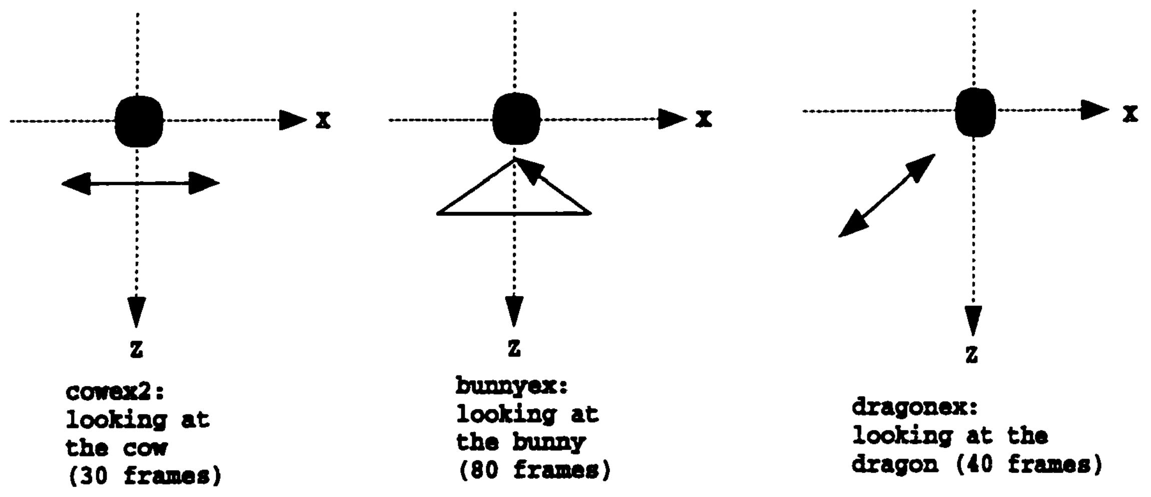


Figure 9.1: Navigation paths in benchmark cowex2, bunnyex and dragonex

- Jade 2.0: It simplifies a triangulated model by removing vertices and re-triangulating the patches using edge flipping. It was developed in the Visual Computer Group of CNUCE/IEI-C.N.R. [6].

These algorithms were all proposed for producing multi-resolution models for real-time rendering. They are used to produce three sets of level of details. However, the LOD selection mechanism is not defined in the algorithms. During real-time rendering, the same distance threshold is used as a criteria to choose level of detail.

Our goal is to test how the three LOD algorithms perform in a set of real-time rendering benchmarks and which one is better than others.

9.1.2 Experiment setup and parameters

Six benchmarks are used in the experiments. Benchmark *headex1* and *headex2* are the same as benchmark 1 and 2 respectively, which were presented in chapter 8. Benchmarks *cowex1* and *cowex2* are navigating a "cow" model shown in figure 5.2(a), along paths *ex1* and *cowex2path*. Benchmark *bunnyex* uses the "Stanford Bunny" model and path *bunnypath*. Benchmark *dragonex* navigates a "dragon" model shown in figure 5.6(b), along path *dragonpath*. Figure 9.1 illustrates the paths in benchmarks *cowex2*, *bunnyex*, *dragonex*

Algorithm	$\bar{f}(msec)$	$minf$ (msec)	$maxf$ (msec)	F	Es	Et
ClusterLOD	19.1167	2	57	18.2772	43.772	5.22969
QslimLOD	17.9667	1	47	17.5589	17.8928	4.38768
JadeLOD	16.4833	1	47	15.8294	17.5531	3.78405
Regular	145.383	145	156	0.613332	0	0

Algorithm	Preprocessing time	Process Memory (kb)	Model Size(face)
ClusterLOD	Manual	5656	4356 + 1052 + 191 + 26
QslimLOD	Manual	5656	4356 + 1042 + 192 + 25
JadeLOD	Manual	5672	4356 + 1050 + 190 + 26
Regular	0	5632	4356

Table 9.1: Experiment results of benchmark headexl, 113 critical vertices

9.1.3 Experiment results

Tables 9.1, 9.2, 9.3, 9.4, 9.5 and 9.6 present the results obtained with RRB. LOD model size in each algorithm is also included in the tables.

The results in all the six experiments(figure 9.2 and 9.3) show that algorithm JadeLOD and QslimLOD produce significantly better spatial image fidelity than algorithm ClusterLOD. Algorithm JadeLOD is the best for preserving spatial image fidelity. It also gives the best temporal image fidelity performance as well. Algorithm QslimLOD ranks the second for temporal image fidelity in the experiments. ClusterLOD produces the worst image fidelity in both the spatial and temporal domains. The last two benchmarks use a very large original model. Three algorithms are designed to render coarse levels of details only, which have 25%, 5%, 1% and 0.5% of the original faces respectively. Thus, they all improve frame rate significantly and use much less memory.

The three algorithms produce similar frame rate performance and memory consumption because the LOD selection criteria is the same for all of them and their model size are almost the same.

In the experiments, the LOD models are generated off-line with the three algorithms. The simplification time is not reported as it is not related to the performance in real-time rendering.

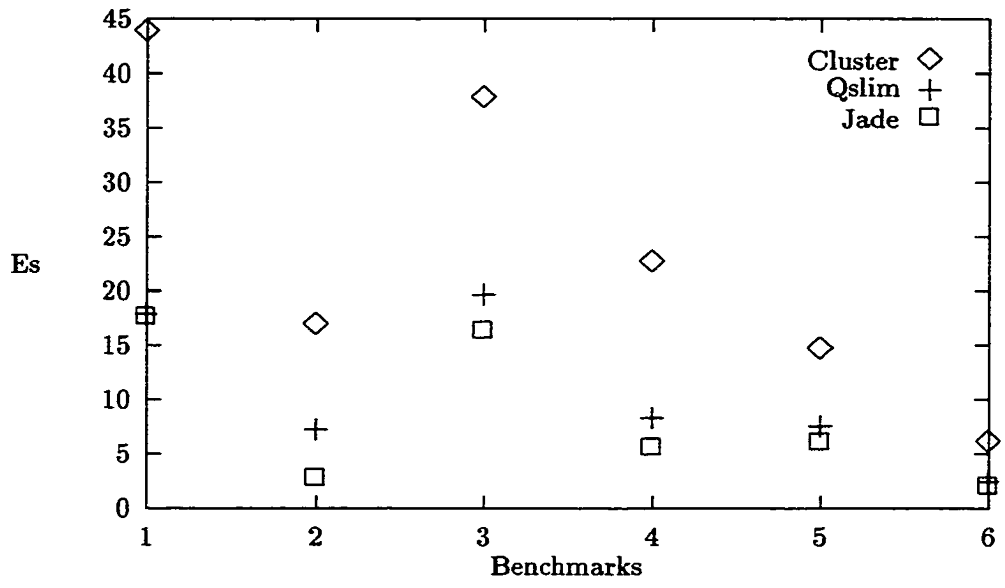


Figure 9.2: Es produced by the three LOD algorithms in six experiments

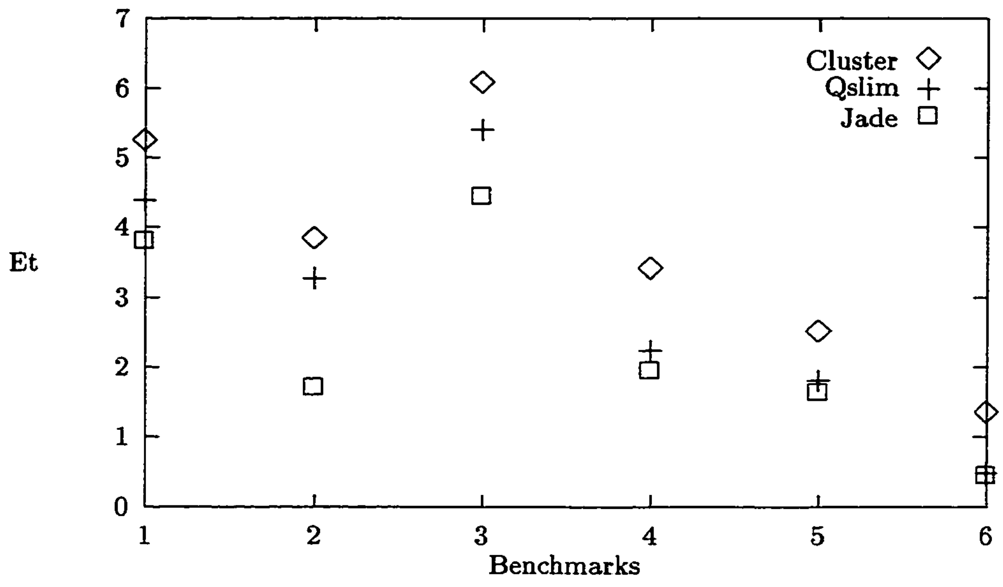


Figure 9.3: Et produced by the three LOD algorithms in six experiments

Algorithm	$\bar{f}$ (msec)	$minf$ (msec)	$maxf$ (msec)	F	Es	Et
ClusterLOD	41.92	38	56	2.6559	16.8642	3.81719
QslimLOD	40.415	37	65	3.37745	7.23394	3.27359
JadeLOD	41.84	36	58	5.1	2.66828	1.68281
Regular	146.27	145	150	0.439303	0	0

Algorithm	Preprocessing time	Process Memory (kb)	Model Size (face)
ClusterLOD	Manual	5696	4356 + 1052 + 191 + 26
QslimLOD	Manual	5700	4356 + 1042 + 192 + 25
JadeLOD	Manual	5656	4356 + 1050 + 190 + 26
Regular	0	5632	4356

Table 9.2: Experiment results of benchmark headex2, 113 critical vertices

Algorithm	$\bar{f}$ (msec)	$minf$ (msec)	$maxf$ (msec)	F	Es	Et
ClusterLOD	26.2833	2	61	20.8306	37.6945	6.06652
QslimLOD	24.8833	2	55	20.7305	19.6469	5.40293
JadeLOD	24.9833	2	54	20.5805	16.2764	4.42467
Regular	192.8	192	214	1.28	0	0

Algorithm	Preprocessing time	Process Memory (kb)	Model Size (face)
ClusterLOD	Manual	6065	5804 + 1450 + 284 + 56
QslimLOD	Manual	6064	5804 + 1444 + 282 + 52
JadeLOD	Manual	6065	5804 + 1444 + 282 + 54
Regular	0	5952	5804

Table 9.3: Experiment results of benchmark cowex1, 621 critical vertices

Algorithm	$\bar{f}$ (msec)	$minf$ (msec)	$maxf$ (msec)	F	Es	Et
ClusterLOD	50.8	49	68	1.85333	22.5632	3.39001
QslimLOD	50.2	49	53	1.2	8.28042	2.23856
JadeLOD	49.433	49	56	0.664445	5.46574	1.92486
Regular	197.167	196	218	1.55556	0	0

Algorithm	Preprocessing time	Process Memory (kb)	Model Size (face)
ClusterLOD	Manual	6084	5804 + 1450 + 284 + 56
QslimLOD	Manual	6104	5804 + 1444 + 282 + 52
JadeLOD	Manual	6100	5804 + 1444 + 282 + 54
Regular	0	5984	5804

Table 9.4: Experiment results of benchmark cowex2, 621 critical vertices

Algorithm	$\bar{f}$ (msec)	$minf$ (msec)	$maxf$ (msec)	F	Es	Et
ClusterLOD	176.225	10	606	169.835	14.6005	2.49024
QslimLOD	185.425	10	620	177.869	7.51062	1.80495
JadeLOD	181.925	10	603	176.732	5.9761	1.61096
Regular	2461.3	2451	2619	13.4575	0	0

Algorithm	Preprocessing time	Process Memory (kb)	Model Size (face)
ClusterLOD	Manual	8808	17200 + 3347 + 636 + 302
QslimLOD	Manual	8852	17200 + 3353 + 633 + 300
JadeLOD	Manual	8832	17200 + 3353 + 634 + 301
Regular	0	21328	69451

Table 9.5: Experiment results of benchmark bunnyex, 359 critical vertices

Algorithm	$\bar{f}$ (msec)	$minf$ (msec)	$maxf$ (msec)	F	Es	Et
ClusterLOD	236.175	17	443	172.601	6.00743	1.32515
QslimLOD	240.625	17	475	175.719	2.39099	0.472434
JadeLOD	236.375	18	446	172.731	1.90055	0.421805
Regular	1753.65	1742	1971	17.27	0	0

Algorithm	Preprocessing time	Process Memory (kb)	Model Size(face)
ClusterLOD	Manual	7676	12688+ 2494 + 470 + 234
QslimLOD	Manual	7704	12638 + 2482 + 476 + 226
JadeLOD	Manual	7692	12638 + 2482 + 476 + 226
Regular	0	16756	50761

Table 9.6: Experiment results of benchmark dragonex, 293 critical vertices

9.1.4 Section summary

RRB is used to measure three LOD algorithms which have never been evaluated in real-time rendering. It produces consistent performance results which make it possible for us to evaluate and compare them. For this reason, we have achieved significant progress towards our ultimate goal for performance evaluation and comparison of multi-resolution algorithms for real-time rendering.

9.2 Measuring other multi-resolution algorithms with RRB

Due to the limitation of resource and time, and also the limited access to the code of current multi-resolution algorithms, other multi-resolution algorithms were not evaluated in this research. However, as we have shown in chapter 7, it is easy to interface a multi-resolution algorithm with RRB through the small and clean interface and evaluate it with RRB.

Currently many view dependent algorithms have been proposed. RRB is also capable of interfacing with these algorithms and testing them. One needs to add a wrapper for the transformation/group node, disable the default traversal mechanism and enable a view culling mechanism.

9.3 Real-time rendering problems

As we discussed earlier, multi-resolution algorithms are one type of real-time rendering algorithm. Other real-time rendering algorithms, such as culling algorithms, may have similar performance problems while they try to speed up rendering. RRB can also measure such algorithms. The algorithms just need to be interfaced with RRB at the proper points and tested in the same way. RRB can help to evaluate the algorithm's performance and also identify the bottlenecks in the real-time rendering pipeline.

9.4 Chapter summary

While RRB was designed as the first prototype system to investigate the performance evaluation problem of multi-resolution algorithms for real-time rendering, it has been successfully used for measuring and comparing current level of detail algorithms. It also shows great potential as a tool for investigating the real-time rendering problem in general.

Chapter 10

Conclusions

This thesis presents a performance evaluation framework for multi-resolution algorithms for real-time rendering. The empirical tests demonstrate that this framework provides a standard measurement environment for multi-resolution algorithms. It is capable of conducting broad and thorough performance tests and producing consistent and meaningful performance results. The performance evaluation framework has been used to effectively measure and compare three typical level of detail algorithms which have never been evaluated in real-time rendering applications.

10.1 Summary of contributions

To review, the primary contributions of our work as described in this thesis are:

- Key performance metrics for multi-resolution algorithms in real-time rendering are defined. They are based on the general requirements that the algorithms are expected to satisfy.

- Novel automatic performance measures of frame rate, spatial image fidelity, and temporal image fidelity are developed. These measures are able to capture typical artifacts produced by algorithms and compute how well algorithms achieve the ideal situation. They are specific, realizable and easy to interpret.

- A testbed and techniques for composing and running real-time rendering benchmarks of various types in a standard environment are developed. They avoid the tedious job of building the tasks and make performance measurements and comparison possible.

- Using the framework, several algorithms are measured in various real-time rendering benchmarks. The measurement results are used to compare their performance.

10.2 Future directions

There are several ways in which this work could be improved and extended in the future. The following avenues appear particularly important or promising.

Improved Image Fidelity Measures

Image fidelity is one of the important issues for evaluating multi-resolution algorithms. In this framework, we measure an algorithm's image fidelity in the spatial and temporal domains. The measures are based on intensity differences of local areas of interest in the original image sequence and test image sequence. The local areas are chosen around the neighborhood of extrema of curvature and surface discontinuities of 3D objects. Experimental results show that they effectively estimate the image distortions produced by multi-resolution algorithms. However, these measures have not been formally correlated with subjective tests. It is not clear if the measures properly capture the just noticeable distortions from visual perception point of view. Formal user studies are needed to answer this question.

As discussed in chapter 5, the results produced by the image fidelity measures, especially temporal image fidelity measure Et, have a small amount of noise caused by the intrinsic aliasing problem in computer graphics. In our experiments, the noise does not affect the image fidelity rankings of test algorithms. However, we believe the noise should be reduced to a minimal level for more accurate measurement in the future. Super sampling or other anti-aliasing techniques can be appropriately applied.

Performance Parameter Analysis

Performance parameter analysis is important in performance comparison. In RRB, task parameters should be thoroughly studied for more effective performance analysis and comparison of algorithms. As we have seen in the experiments, the task parameters affect an algorithm's performance. In the framework, we suggested to evaluate an algorithm based on multiple measures with different benchmarks for fair results.

We leave the parameter analysis and performance comparison to future study, which are believed to be a very interesting project.

Extending the testbed and broader range of benchmarks

Our experience with multi-resolution algorithms shows they perform differently on different tasks. To fairly evaluate and compare their performance, various real-time rendering tasks are needed.

In the framework, the testbed is designed to load and run 3D navigation tasks as real-time rendering benchmarks. These tasks are composed of virtual environments written in the VRML 2.0 format and navigation paths. Due to limited resources, we are unable to incorporate animation in the scene graph of the current prototype system. Our experience with RRB indicates that it is an important extension. Many interesting virtual environments obtained from the Internet consist of animated objects. While loading such tasks, RRB has to ignore the animation feature and change the associated objects to static ones as it doesn't have animation support. Thus the number of benchmark options is greatly reduced. Given the current VRML 2.0 parser and the scene graph architecture, incorporating basic animation support in RRB is not technically difficult. It can be done by implementing the VRML 2.0 event model.

3D navigation tasks covers a large class of real-time rendering tasks used in practice. However, there are many other interactive 3D applications where multi-resolution algorithms should be tested in as well. For example, real-time CAD/CAM tasks often need multi-resolution algorithms to give the designer real-time response. Such tasks involve more complex interactions between a user and a CAD/CAM environment. Often a user wants to have the capability to manipulate single objects directly in a CAD/CAM environment. The current virtual environment and navigation path model in RRB is certainly not sufficient to satisfy this requirement. In interactive 3D graphics, navigation is considered one of the simplest interactions and is easy to develop. CAD/CAM tasks require far more complex interaction techniques. Liang reports a good investigation in this area [38]. When measuring real-time rendering algorithms with CAD/CAM tasks, one is not only concerned with developing interaction methods, but more importantly concerned with providing a technique for users to conveniently load, run and replay these CAD/CAM tasks. Thus various algorithms can be measured and compared in a unified environment. Extending RRB in this direction is considered very challenging and useful.

Measurement and Comparison of Broader Range of Algorithms with RRB

The performance evaluation framework was developed to evaluate and compare multi-resolution algorithms. As a set of performance evaluation tools, RRB greatly reduces the effort of performance measurement of multi-resolution algorithms in real-time rendering and makes the evaluation and comparison possible. In this thesis, we describe our experiences of evaluating Lau's simplification list and its variants. It shows interfacing these algorithms with RRB is quite convenient and clean given the flexible and well design scene graph architecture. Most continuous multi-resolution algorithms, such as [39][18][30][23], are similar to the simplification list [34]. Given a C++ implementation of the algorithms, one just needs to write wrapper classes to bridge the prototype system and the algorithms at the IndexedFaceSet node. Then, performance measurements can be done in a few steps using RRB. In the thesis we also reported evaluating three LOD algorithms in RRB. LOD is one of the basic multi-resolution techniques. RRB integrates the LOD mechanism in its implementation. We have shown it can evaluate such algorithms directly given their LOD representations in a VRML 2.0 file, In all the experiments, the measurement results produced by RRB explicitly show which algorithm performs better than others and under which conditions an algorithm behaves best.

Many other multi-resolution algorithms or ideas have been proposed for real-time rendering. However, they have never been evaluated and compared. It would be beneficial to both multi-resolution and real-time rendering community if more algorithms are evaluated and compared with RRB. Recently, view-dependent multi-resolution algorithms, such as [41][65][30], have drawn increasing attention in the multi-resolution community. They are believed to be good ideas for real-time rendering problems. However, their performance should be evaluated and compared to show their potential and limitation. With the flexible scene graph architecture, RRB allows these algorithms to be interfaced as well. As view culling techniques are involved in these algorithms, they will need to attach RRB at higher level nodes, which are the Group, Transform, LOD and Collision nodes. If an algorithm attaches RRB at a higher level node, it has more flexibility to manipulate the scene graph and also has more responsibility in the mean time. In this case, the algorithms take the responsibility of implementing both view culling and level of detail control. Again wrapper classes are required to bridge the algorithms' implementation and RRB. Hierarchical image cache is another variant of multi-resolution algorithms. The idea is to hierarchically model a 3D scene and store it in image caches. During real-time rendering, the cached images of far away objects are reused to improve speed. Currently RRB doesn't support image caches in the scene graph nodes. To evaluate such algorithms, caching mechanism needs to be incorporated. Another solution is to interface the algorithm with RRB at the scene graph root level and let the algorithm itself maintain its own image caches.

RRB is indeed a real-time rendering framework. If we look at real-time rendering algorithms in general, most of them have similar performance problems as multi-resolution algorithms, which include frame rate, image fidelity, resource consumption, etc. The techniques developed in this thesis should be extended to measure many of the real-time rendering algorithms. It can help to find which algorithms perform better than others in a standard real-time rendering environment. On the other hand, it facilitates the identification of problems and limitations of current techniques, and thus inspires improvements in them. We believe that performance evaluation and comparison of more algorithms with RRB is the most promising avenue for improving multi-resolution technologies and also other real-time rendering techniques in general.

Bibliography

[1] John M. Airey, John H. Rohlf, and Frederick P. Brooks. Towards Image Realism with Interactive Update Rate In Complex Virtual Building Environments. In *Computer Graphics(1990 Symposium on Interactive 3D Graphics*, pages 41–50, March 1990.

[2] Daniel G. Aliaga. Visualization of Complex Models Using Dynamic Texture-based Simplification. In *IEEE Visualization '96*, pages 101–106, April 1996.

[3] F. Attneave. Some Informational Aspects of Visual Perception . *Psychological Review*, 61:183–193, 1954.

[4] Rikk Carey and Gavin Bell. *The Annotated VRML97 Reference Manual.* Addison-Wesley, 1997.

[5] Bradford Chamberlain, Tony DeRose, Dani Lischineski, David Salesin, and John Snyder. Fast Rendering of Complex Environments Using a Spatial Hierarchy. In *Proceedings of Graphics Interface'96*, May 1996.

[6] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13(5):228–246, 1997.

[7] P. Cignoni, R. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. Technical report, Istituto per l'Elaborazione dell'Infomazione - Consiglio Nazionale delle Ricerche, 1997.

[8] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: measuring error on simplified surfaces. Technical report, Istituto per l'Elaborazione dell'Infomazione - Consiglio Nazionale delle Ricerche, 1996.

[9] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-Preserving Simplification. In *Proc. ACM SIGGRAPH '98*, pages 115–122, July 1998.

[10] Vrml Consortium. The Virtual Reality Modeling Language. *http://www.vrml.org*, 1995.

[11] Web3D Consortium. The Virtual Reality Modeling Lauguage Repository. *http://www.web3d.org/vrml/vrml.htm*, 1999.

[12] Patrick Courtney, Neil Thacker, and Adrian F. Clark. Algorithmic Modelling for Performance Evaluation . *Machine Vision and Applications*, 9(5-6):219–228, 1997.

[13] Rudy P. Darken and John L. Sibert. A toolset for navigation in virtual environments. In *Proc. ACM Symposium on User Interface Software and Technology '93*, pages 157–265, November 1993.

[14] K. Dezhgosha, M.M Janali, and S.C. Kwatra. Performance Evaluation of the VQ based Image Coding Algorithm. In *1990 IEEE International Symposium on Circuits and Systems, part 4*, pages 3057–3060, May 1990.

[15] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. In *IEEE Visualization '97*, pages 81–88, October 1997.

[16] James A. Ferwerda, Sumanta N. Pattnaik, Peter Shirley, and Donald P. Greenberg. A Model of Visual Masking for Computer Graphics. In *Proc. ACM SIGGRAPH '97*, pages 143–152, August 1997.

[17] Leila De Floriani, Paola Magillo, and Enrico Puppo. Efficient implementation of multi-triangulations. In *IEEE Visualization '98*, pages 43–50, October 1998.

[18] Leila De Floriani, Enrico Poppo, and Paola Magillo. *Geometric Modeling: Theory and Practice*, chapter A formal approach to multiresolution hypersurface modeling. Springer Verlag, 1997.

[19] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics Principles and Practice.* Addison-Wesley Publishing Company, 2 edition, 1990.

[20] Aubrey R. Jr. Fowler and Stephen C. Bushardt. T.O.P.E.S. : Developing a Task Oriented Performance Evaluation System. *SAM Advanced Management Journal*, 51(4):4–8, 1986.

[21] Thomas A. Funkhouser and Carlo H. Sequin. Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments. In *Proc. ACM SIGGRAPH '93*, pages 247–255, August 1993.

[22] M. Garland and P. Heckbert. Surface Simplification using Quadric Error Metrics. In *Proc. ACM SIGGRAPH '97*, pages 209–216, August 1997.

[23] Michael Garland. *Quadric-Based Polygonal Surface Simplification.* PhD thesis, Carnegie Mellon University, May 1999.

[24] Rafael C. Gonzalez and Paul Wintz. *Digital Images Processing* . Addison-Wesley Publishing Company, 1987.

[25] Mark Green. A Framework for Real-Time Rendering in Virtual Reality. In *ACM VRST'96*, pages 3–9, July 1996.

[26] D.G. Haack. *Statistical Literacy: A Guide to Interpretation.* Duxbury Press, 1981.

[27] P.S. Heckbert and M. Garland. Survey of Polygonal Surface Simplification Algorithms. In *SIGGRAPH '97 Course Notes*, August 1997.

[28] Huges Hoppe. Progressive meshes. In *Proc. ACM SIGGRAPH '96*, pages 99–108, August 1996.

[29] Huges Hoppe. View-Dependent Refinement of Progressive meshes. In *Proc. ACM SIGGRAPH '97*, pages 189–198, August 1997.

[30] Huges Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *IEEE Visualization '98*, pages 35–42, October 1998.

[31] Veysi Isler, Rynson Lau, and Mark Green. Real-time Multi-resolution Modeling for Complex Virtual Environments. In *ACM VRST'96*, pages 11–20, July 1996.

[32] Raj Jain. *The Art of Computer Systems Performance Analysis.* John Wiley &Sons, INC., 1991.

[33] R.S. King and B. Julstrom. *Applied Statistic Using the Computer* . Mayfield Publishing, 1982.

[34] Rynson W.H. Lau, Mark Green, Danny To, and Janis Wong. Real-Time Continuous Multi-Resolution Method for Models of Arbitrary Topology. *Presence: Teleoperators and Virtual Environments, MIT Press*, 7(1):22–35, February 1998.

[35] Daniel Lauzon, Andre Vincent, and Limin Wang. Performance Evaluation of MPEG-2 Video Coding for HDTV . *IEEE Transaction on Broadcasting*, 42(2):88–94, 1996.

[36] S. Lavenberg. *Computer Performance Modeling Handbook.* Academic Press, New York, 1983.

[37] R.I. Levin. *Statistics for Management* . Prentice-Hall, 1981.

[38] Jiandong Liang. *Interaction Techniques for Solid Modeling with a 3D Input Device.* PhD thesis, University of Alberta, September 1995.

[39] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner. Real-time, continuous level of detail rendering of height fields. In *Proc. ACM SIGGRAPH '96*, pages 109–118, August 1996.

[40] Peter Lindstrom and Greg Turk. Fast and Memory Efficient Polygonal Simplification. In *IEEE Visualization '98*, 1998.

[41] David Luebke and Carl Erikson. View-Dependent Simplification of Arbitrary Polygonal Environment. In *Proc. ACM SIGGRAPH '97*, pages 199–208, August 1997.

[42] P.W.C. Maciel and P. Shirley. Visual Navigation of Large Environments Using Textured Clusters. In *1995 Symposium on Interactive 3D Graphics*, pages 95–102, May 1995.

[43] Tomas Moller and Eric Haines. *Real-Time Rendering.* A K Peters Ltd., 1999.

[44] OpenGL Performance Characterization Organization. The OpenGL Performance Characterization Project . *http://www.specbench.org/gpc/opc.static/index.html*, 1998.

[45] Matthew Regan and Ronald Pose. Priority Rendering with a Virtual reality Address Recalculation Pipeline. In *Proc. ACM SIGGRAPH '94*, pages 155–162, August 1994.

[46] John Rohlf and James Helman. IRIS Performer: A high performance multiprocessing toolkit for real-time 3D graphics. In *Proc. ACM SIGGRAPH '94*, pages 381–394, August 1994.

[47] Jarek Rossignac and Paul Borrel. Multi-resolution 3D Approximations for Rendering Complex Scenes. Technical report, IBM, 1992.

[48] Gernot Schaufler and Wofgang Sturzlinger. A Three Dimensional Image Cache for Virtual Reality. In *Proceedings of Eurographics'96*, pages 227–248, April 1996.

[49] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of Triangle Meshes. In *Proc. ACM SIGGRAPH '92*, pages 65–69, July 1992.

[50] Jonathan Shade, Dani Lischinski, David H. Salesin, Tony DeRose, and John Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In *Proc. ACM SIGGRAPH '96*, pages 75–82, August 1996.

[51] Chris D. Shaw, Mark Green, Jiandong Liang, and Yunqi Sun. Decoupled Simulation in Virtual Reality with the MR Toolkit. *ACM Transactions on Information Systems*, 11(3):287–317, July 1993.

[52] Anthony Steed. Efficient navigation around complex virtual environments . In *ACM VRST'97*, pages 173–180, September 1997.

[53] Steve E. Tice, Mike Fusco, and Paul Straley. The Picture Level Benchmark. *Computer Graphics World*, pages 123–130, July 1989.

[54] Steve Upstill. *The Renderman Companion.* Addison-Wesley Publishing Company, 1990.

[55] Christian J. van den Branden Lambrecht. Automatically Assessing MPEG Coding Fidelity. *IEEE Design and Test of Computers*, 12(4):28–33, 1995.

[56] Christian J. van den Branden Lambrecht. A Working Spatio-Temporal Model of the Human Visual System for Image Restoration and Quality Assessment Applications. In *Proceedings of ICASSP96*, 1996.

[57] Norman G. Vinson. Design guidelines for landmarks to support navigation in virtual environments. In *Proc. CHI '99 Conference on Human Factors in Computing Systems*, pages 278–285, May 1999.

[58] Stephen Voran and Stephen Wolf. The Development and Correlation of Ojbective and Subjective Video Quality Measures. In *IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing'91*, pages 483–485, May 1991.

[59] Andrew B. Watson. *Digital Images and Human vision.* The MIT Press, 1993.

[60] Benjamin Watson, Victoria Spaulding, Neff Walker, and William Ribarsky. Evaluation of the Effects of Frame Time Variation on VR Task Performance . Technical report, Georgia Institute of Technology, 1996.

[61] Josie Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D Graphics With Open Inventor, Release 2.* Addison-Wesley Publishing Company, 1994.

[62] Stephen Wolf, Margaret Pinson, Stephen Voran, and Arthur Webster. Objective Quality Assessment of Digitally Transmitted Video. In *IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing'91*, pages 477–482, May 1991.

[63] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide, Second Edition.* Addison-Wesley Developers Press, 1997.

[64] Cosmo Worlds. Cosmo Worlds 2.0 Technical Specifications. *http://www.cosmosoftware.com/products/worlds/*, 1999.

[65] Julie C. Xia and Amitabh Varshney. Dynamic View-Dependent Simplification for Polygonal Models. In *IEEE Visualization '96*, pages 327–334, 1996.

[66] Douglas A. Young. *Object Oriented Programming with C++ and OSF/Motif 2nd Edition.* Prentice Hall PTR, 1995.

[67] Ping Yuan, Mark Green, and Rynson Lau. A Framework of Performance Evaluation of Real-time Rendering Algorithms in Virtual Reality. In *ACM VRST'97*, pages 51–59, September 1997.

[68] Ping Yuan, Mark Green, and Rynson Lau. Benchmarking Mesh Simplification Algorithms in Real-time Rendering Applications. In *Proceedings of the Tenth Western Computer Graphics Symposium*, 1999.

[69] William Y. Zou. Digital HDTV Compression Techniques for Terrestrial Broadcasting. *SMPTE Journal*, pages 127–131, February 1993.

[70] William Y. Zou. Performance Evaluation: From NTSC to Digitally Compressed Video. *SMPTE Journal*, pages 795–800, December 1994.

[71] Micheal J. Zyda, Mark A. Fichten, and David H. Jennings. Meaningful Graphics Workstation Performance Measurements. *Computers and Graphics*, 14(3):519–526, 1990.

Appendix A

RRB Implementation Notes

The purpose of this appendix is to augment the outline of the prototype implementation. We focus on some key points, leaving out various support functions and details. RRB is developed as an object-oriented application framework using C++ with Standard Template Library, OpenGL, Motif object oriented framework, Flex and Bison. Figure 7.1 shows the main classes that form its architecture. It is designed to be portable over a wide range of platforms and to be extended easily.

A.1 Virtual Environment

Virtual environment is one of the core component of RRB. It is built around a VRML 2.0 parser and a set of scene graph nodes which are used to construct the scene graph hierarchy. Two primary concerns have motivated the design of this component. First, it is important that RRB loads various available virtual environment datasets in the VRML 2.0 format. Thus, a wide range of real-time rendering benchmarks can be automatically generated to measure multi-resolution algorithms. Second, RRB is meant to be reasonably efficient in its use of memory. The scene graph is expected to be self-complete and contain only a minimal set of nodes that are necessary for multi-resolution algorithms to perform real-time rendering tasks. The VRML 2.0 specification defines a large collection of nodes to describe 3D graphics for the web. Many of them are not concerned for multi-resolution algorithms at the moment, such as Anchor, Audioclip, and etc.. If these nodes appear in a VRML file, they need to be filtered out when the scene graph is built.

A.1.1 The VRML 2.0 parser

In RRB, the VRML 2.0 parser is responsible for parsing a VRML 2.0 file, and generating the in-memory scene graph as described in chapter 6. It is designed based on the prototype nodes defined in the VRML 2.0 specification and developed with flex and and bison. Our parser is capable of parsing any VRML 2.0 file. However, as some of these nodes are not of interests of RRB, they are not built into the scene graph when they are scanned. For example, if a file contains an *Anchor* node which is not defined in our scene graph, the parser still parses this node and determines if it follows the VRML 2.0 syntax. If it does, the parser will go on to parse other nodes without instancing this node and its children into memory. If it does not follow the VRML 2.0 syntax, the parser will report a syntax error and exit. When these nodes are successfully scanned by the parser, their instances will be loaded into the memory to build a scene graph. For example, if an *IndexedFaceSet* node is scanned successfully, an instance of the IndexedFaceSet class is created and loaded in the memory as it is defined in our scene graph. If parsing is failed, an error is reported and the program exits.

A.1.2 The SceneGraph Nodes

As described in the previous chapters, a scene graph is a direct acyclic graph (DAG). It consists of the root node, the internal nodes and the leaf nodes. The SceneGraph class implements the root of a scene graph. It maintains a list of scene graph node objects which can be internal nodes or leaf nodes and provides the basic traversal methods. The internal nodes include Collision, Group, LOD and Transform. They all contain an object of the MFNode class which maintains a list of scene graph nodes as their children. The internal nodes provide their own implementation of the list traversal and rendering methods. The rest are the leaf nodes which define their own attributes and methods. All the internal and leaf nodes are derived from a super class called SFNode. The SFNode class defined the common attributes of these node classes. It also defines a set of virtual methods that the node classes must implement, such as render. The SceneGraphObj class is the root of the entire class inheritance hierarchy. It defines the common attributes of all the scene graph objects, such as

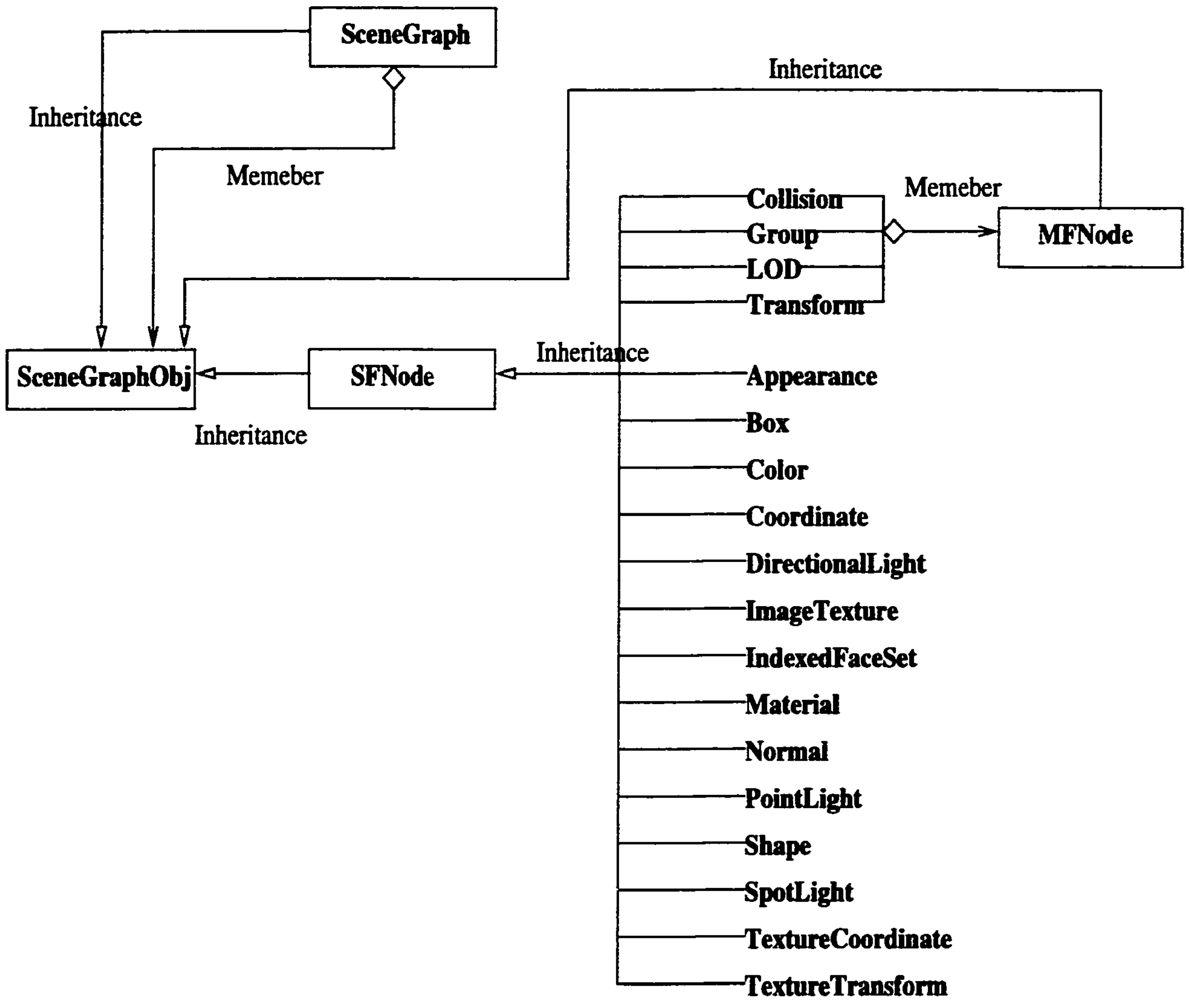


Figure A.1: Scene graph node class hierarchy

their name, type, etc.. Every other class is derived from the SceneGraphObj class. Figure A.1 illustrates the class inheritance hierarchy in the RRB scene graph.

A.2 Navigation Path

A.2.1 The EyePath class

A navigation path simulates a set of viewpoints and view orientations which the user uses to navigate a virtual environment. In RRB, it is represented by an object of EyePath class. EyePath class defines the data structure of a path as a set of 3D coordinates and quaternions. It also defines two key behaviors for its objects:

- Update the viewpoint and view orientation for the current frame
- Draw the scene according to the viewpoint and view orientation

These two behaviors are defined as pure virtual functions and must be implemented in the subclasses of the EyePath class.

A.2.2 The subclasses

As described in the previous chapters, RRB supports both level 1 path and level 2 path. Eyepath defines a general interface for the navigation path. The implementation are encapsulated in the subclasses of the EyePath class. They are MathPath, TrackerPath and DataPath. MathPaths implements a navigation path which are mathematically generated based on some key interpolation points. TrackerPath defines a navigation scheme in which the navigation path is determined by a 3D tracker device. The device's position, orientation and the button press actions performed on the device are used to obtain the navigation path. DataPath reuses the path data generated and recorded by the TrackerPath.

A.2.3 Measures

The measures of frame rate, Es, and Et are mainly implemented in three classes. They are the Timer class, FrameBuffer class, and DbenchMark class.

The Timer class

The Timer class implements a mechanism to obtain the timing of a block of function calls and maintains a buffer of the timing data so that the frame rate performance can be computed later.

The FrameBuffer class

The FrameBuffer class implements a set of methods to obtain the RGB value of an array of pixels in the rendered image for each frame. These values are used to compute the average intensity data.

The DbenchMark class

The DbenchMark class implements methods to automatically select critical vertices given a scene graph. The critical vertices are then projected to the frame buffer, according to the viewpoint and view orientation, to obtain the local areas of interests. It is the connection point of the Ve class and the FrameBuffer class and also the core of the image fidelity measures.

A.3 RRB bootstraps

A.3.1 The RRBWorkspace class

The RRBWorkspace class is the central controller of RRB. It maintains global properties and references to the objects of Ve, EyePath, DbenchMark, Timer and FrameBuffer. It provides bootstraps to load a virtual environment, choose an algorithm, perform a set of measurement operations and dump measurement results.

A.3.2 RRB graphical user interface

As shown in figure 7.3, RRB is provided with a graphical user interface for user's convenience. To simplify the development effort, its implementation makes use of the MotifApp application framework [66], The MotifApp application framework provides a collection of user interface components, such as Application, MainWindow, Cmd, MenuBar, Button and etc. It also encapsulates a basic structure of applications based on X and Motif. The graphical user interface of RRB is implemented by reusing these components and the basic application structure defined in the MotifApp framework. The RRBApp class is the central point of the implementation. It derives from the central class — Application which is defined in the MotifApp application framework. RRBApp is responsible for managing the resources of the RRBController window and the RRBView window. It also registers the commands and invokes callback functions when events occur.